Package 'genoset'

April 5, 2014

Type Package

Title Provides classes similar to ExpressionSet for copy number analysis

Version 1.14.0

Date 2011-01-15

Author Peter M. Haverty

Maintainer Peter M. Haverty <phaverty@gene.com>

Description Load, manipulate, and plot copynumber and BAF data. GenoSet class extends eSet by adding a ``locData'' slot for a RangedData or GRanegs object. This object contains feature genome location data and provides for efficient subsetting on genome location. CNSet and BAFSet extend GenoSet and require assayData matrices for Copy Number (cn) or Log-R Ratio (lrr) and B-Allele Frequency (baf) data. Implements and provides convenience functions for processing of copy number and B-Allele Frequency data.

License Artistic-2.0

LazyLoad yes

Depends R (>= 2.10), BiocGenerics (>= 0.1.6), Biobase (>= 2.15.1),IRanges, GenomicRanges

Imports methods, graphics, IRanges, GenomicRanges

Suggests RUnit, DNAcopy, stats, BSgenome, Biostrings

Enhances parallel

biocViews Infrastructure, DataRepresentation, Microarray, SNP,CopyNumberVariants

Collate 'genoset-class.R' 'cnset-class.R' 'bafset-class.R' 'DataFrame-methods.R' 'test_genoset_package.R' 'utils.R' 'ordering.R' 'segments.R' 'bounds.R' 'plots.R'

ByteCompile TRUE

R topics documented:

`genoset-package` *GenoSet: An eSet for data with genome locations*

Description

Load, manipulate, and plot copynumber and BAF data. GenoSet class extends eSet by adding a "locData" slot for a GenomicRanges object. This object contains feature genome location data and provides for efficient subsetting on genome location. Genoset also implements an number of convenience functions for processing of copy number and B-Allele Frequency data and for working with segmented data.

See Also

genoset-datasets GenoSet

`baf` *Get baf data*

Description

Get or Set the baf assayData slot

Arguments

`object`	A BAFset object

Details

Get or Set the baf assayData slot

Value

matrix

Author(s)

Peter M. Haverty

`baf2mbaf` *Calculate mBAF from BAF*

Description

Calculate Mirrored B-Allele Frequence (mBAF) from B-Allele Frequency (BAF) as in Staaf et al., Genome Biology, 2008. BAF is converted to mBAF by folding around 0.5 so that is then between 0.5 and 1. HOM value are then made NA to leave only HET values that can be easily segmented. Values > hom.cutoff are made NA. Then, if genotypes (usually from a matched normal) are provided as the matrix 'calls' additional HOMs can be set to NA. The argument 'call.pairs' is used to match columns in 'calls' to columns in 'baf'.

Usage

```
baf2mbaf(baf, hom.cutoff = 0.95, calls = NULL,
  call.pairs = NULL)
```

Arguments

`baf`	numeric matrix of BAF values
`hom.cutoff`	numeric, values above this cutoff to be made NA (considered HOM)
`calls`	matrix of NA, CT, AG, etc. genotypes to select HETs (in normals). Dimnames must match baf matrix.
`call.pairs`	list, names represent target samples for HOMs to set to NA. Values represent columns in "calls" matrix.

Value

numeric matix of mBAF values

Author(s)

Peter M. Haverty

Examples

```
data(genoset)
   mbaf = baf2mbaf( genoset.ds[, , "baf"], hom.cutoff=0.9 )
  calls = matrix(sample(c("AT","AA","CG","GC","AT","GG"),(nrow(genoset.ds) * 2),replace=TRUE),ncol=2,dimnames=li
  mbaf = baf2mbaf( genoset.ds[, , "baf"], hom.cutoff=0.9, calls = calls, call.pairs = list(K="L",L="L") ) # Sample L
  genoset.ds[, ,"mbaf"] = baf2mbaf( genoset.ds[, , "baf"], hom.cutoff=0.9 ) # Put mbaf back into the BAFSet object a
```

`BAFSet` *Create a BAFSet object*

Description

This function is the preferred method for creating a new BAFSet object. Users are generally discouraged from calling "new" directly. This BAFSet function enforces the requirement for "lrr" and "baf" matrices. These and any other "..." arguments will become part of the assayData slot of the resulting object. "..." can be matrices or DataFrame objects (from the IRanges package). This function passes control to the "initGenoSet" method which performs argument checking including dimname matching among relevant slots and sets everything to genome order. Genome order can be disrupted by "[" or "[[" calls and will be checked by methods that require it.

Usage

```
BAFSet(locData, lrr = NULL, baf = NULL, pData = NULL,
  annotation = "", universe, assayData = NULL, ...)
```

Arguments

`locData`	A GRanges or RangedData object specifying feature chromosome locations. featureNames (names or rownames) are required to match featureNames of assayData.
`lrr`	numeric matrix of copy number data with rownames matching featureNames and colnames matching sampleNames
`baf`	numeric matrix of B-Allele Frequency data with rownames matching featureNames and colnames matching sampleNames
`pData`	A data frame with rownames matching all data matrices
`annotation`	character, string to specify chip/platform type
`universe`	character, a string to specify the genome universe for locData. Overrides any universe/genome data in locData.
`assayData`	assayData, usually an environment
`...`	More matrix or DataFrame objects to include in assayData slot

Details

The BAFSet class is defunct. Please use GenoSet. BAFSet only added the baf/lrr getter/setter functions, which are redundant with x[, , 'baf'] and x[, , 'lrr'] now.

Value

A BAFSet object

Author(s)

Peter M. Haverty

See Also

bafset-class, genoset-class

`BAFSet-class` *Class* `"BAFSet"`

Description

A BAFSet is and extension of GenoSet that requires 'baf' and 'lrr' assayData element

Objects from the Class

Objects can be created by calls of the form `new("BAFSet", assayData, phenoData, featureData, experimentData, an` However, as per BioConductor standard practice the object creation function `BAFSet` is recommended.

Slots

`locData`**:** Object of class `"GenomicRangesOrRangedData"` Feature locations on the genome

`assayData`**:** Object of class `"AssayData"` ~~

`phenoData`**:** Object of class `"AnnotatedDataFrame"` ~~

`featureData`**:** Object of class `"AnnotatedDataFrame"` ~~

`experimentData`**:** Object of class `"MIAxE"` ~~

`annotation`**:** Object of class `"character"` ~~

`protocolData`**:** Object of class `"AnnotatedDataFrame"` ~~

`.__classVersion__`**:** Object of class `"Versions"` ~~

Extends

Class `"GenoSet"`, directly. Class `"eSet"`, by class "GenoSet", distance 2.

Methods

show `signature(object = "BAFSet")`: ...

baf `signature(object = "BAFSet")`: Getter for 'baf' assayDataElement

baf<- `signature(object = "BAFSet", value = "matrix")`: setter for 'baf' assayDataElement

genoPlot `signature(x = "BAFSet", y = "ANY")`: Plot data along the genome. Defaults to 'lrr' assayDataElement.

lrr `signature(object = "BAFSet")`: Getter for 'lrr' assayDataElement

lrr<- `signature(object = "BAFSet", value = "matrix")`: Setter for 'lrr' assayDataElement

Author(s)

Peter M. Haverty <`phaverty@gene.com`>

See Also

`BAFSet`, `CNSet`, `GenoSet`

Examples

```
showClass("BAFSet")
```

BAFSet.to.ExpressionSets

Make a pair of ExpressionSets from a BAFSet

Description

Often it is convenient to have a more standard "ExpressionSet" rather than a BAFSet. For example, when using infrastructure dependent on the ExpressionSet slots, like limma or ExpressionSetOnDisk. This will create a list of two ExpressionSets, one each for the baf and lrr data. To make a single ExpressionSet, with the lrr data in the exprs slot and the baf data as an additional member of assayData, use the standard coercion eset = as(bafset,"ExpressionSet").

Usage

```
BAFSet.to.ExpressionSets(bs)
```

Arguments

`bs`	A BAFset object

Details

BAFSEt.toExpressionSets has been defunct. Please use as(x, 'ExpressionSet').

Value

A list with one ExpressionSet each for the baf and lrr data in the BAFSet object

Author(s)

Peter M. Haverty

boundingIndices *Find indices of features bounding a set of chromosome ranges/genes*

Description

This function is similar to findOverlaps but it guarantees at least two features will be covered. This is useful in the case of finding features corresponding to a set of genes. Some genes will fall entirely between two features and thus would not return any ranges with findOverlaps. Specifically, this function will find the indices of the features (first and last) bounding the ends of a range/gene (start and stop) such that first <= start < stop <= last. Equality is necessary so that multiple conversions between indices and genomic positions will not expand with each conversion. Ranges/genes that are outside the range of feature positions will be given the indices of the corresponding first or last index rather than 0 or n + 1 so that genes can always be connected to some data.

Usage

```
boundingIndices(starts, stops, positions,
  valid.indices = TRUE, all.indices = FALSE, offset = 0)
```

Arguments

`starts`	integer vector of first base position of each query range
`stops`	integer vector of last base position of each query range
`positions`	Base positions in which to search
`valid.indices`	logical, TRUE assures that the returned indices don't go off either end of the array, i.e. 0 becomes 1 and n+1 becomes n
`offset`	integer, value to add to all returned indices. For the case where positions represents a portion of some larger array (e.g. a chr in a genome)
`all.indices`	logical, return a list containing full sequence of indices for each query

Details

This function uses some tricks from findIntervals, where is for k queries and n features it is O(k * log(n)) generally and ~O(k) for sorted queries. Therefore will be dramatically faster for sets of query genes that are sorted by start position within each chromosome. The index of the stop position for each gene is found using the left bound from the start of the gene reducing the search space for the stop position somewhat. This function has important differences from boundingIndices2, which uses findInterval: boundingIndices does not check for NAs or unsorted data in the subject positions. Also, the positions are kept as integer, where boundingIndices2 (and findInterval) convert them to doubles. These assumptions are safe for position info coming from a GenoSet, GRanges, or RangedData.

Value

integer matrix of 2 columms for start and stop index of range in data or a list of full sequences of indices for each query (see all.indices argument)

Author(s)

Peter M. Haverty `<phaverty@gene.com>`

See Also

Other "range summaries": `boundingIndices2`, `boundingIndicesByChr`, `rangeColMeans`, `rangeSampleMeans`

Examples

```
starts = seq(10,100,10)
  boundingIndices( starts=starts, stops=starts+5, positions = 1:100 )
```

`boundingIndices2` *Find indices of features bounding a set of chromosome ranges/genes*

Description

This function is similar to findOverlaps but it guarantees at least two features will be covered. This is useful in the case of finding features corresponding to a set of genes. Some genes will fall entirely between two features and thus would not return any ranges with findOverlaps. Specifically, this function will find the indices of the features (first and last) bounding the ends of a range/gene (start and stop) such that first <= start <= stop <= last. Equality is necessary so that multiple conversions between indices and genomic positions will not expand with each conversion. This function uses findIntervals, which is for k queries and n features is O(k * log(n)) generally and ~O(k) for sorted queries. Therefore will be dramatically faster for sets of query genes that are sorted by start position within each chromosome. This should give performance for k genes and n features that is ~O(k) for starts and O(k * log(n)) for stops and ~O(k * log(n)) overall. Ranges/genes that are outside the range of feature positions will be given the indices of the corresponding first or last index rather than 0 or n + 1 so that genes can always be connected to some data.

Usage

```
boundingIndices2(starts, stops, positions, offset = NULL)
```

Arguments

`starts`	numeric or integer, first base position of each query range
`stops`	numeric or integer, last base position of each query range
`positions`	Base positions in which to search
`offset`	integer, value to add to all returned indices. For the case where positions represents a portion of some larger array (e.g. a chr in a genome)

Value

integer matrix of 2 columms for start and stop index of range in data

Author(s)

Peter M. Haverty

See Also

Other "range summaries": `boundingIndices`, `boundingIndicesByChr`, `rangeColMeans`, `rangeSampleMeans`

Examples

```
starts = seq(10,100,10)
  boundingIndices2( starts=starts, stops=starts+5, positions = 1:100 )
```

---

boundingIndicesByChr *Find indices of features bounding a set of chromosome ranges/genes, across chromosomes*

Description

Finds subject ranges corresponding to a set of genes (query ranges), taking chromosome into account. Specifically, this function will find the indices of the features (first and last) bounding the ends of a range/gene (start and stop) such that first <= start < stop <= last. Equality is necessary so that multiple conversions between indices and genomic positions will not expand with each conversion. Ranges/genes that are outside the range of feature positions will be given the indices of the corresponding first or last index on that chromosome, rather than 0 or n + 1 so that genes can always be connected to some data. Checking the left and right bound for equality will tell you when a query is off the end of a chromosome.

Usage

```
boundingIndicesByChr(query, subject)
```

Arguments

`query`	GRanges or something coercible to GRanges
`subject`	RangedData

Details

This function uses some tricks from findIntervals, where is for k queries and n features it is O(k * log(n)) generally and ~O(k) for sorted queries. Therefore will be dramatically faster for sets of query genes that are sorted by start position within each chromosome. The index of the stop position for each gene is found using the left bound from the start of the gene reducing the search space for the stop position somewhat.

This function differs from boundingIndices in that 1. it uses both start and end positions for the subject, and 2. query and subject start and end positions are processed in blocks corresponding to chromosomes.

Both query and subject must be in at least weak genome order (sorted by start within chromosome blocks).

Value

integer matrix with two columns corresponding to indices on left and right bound of queries in subject

Author(s)

Peter M. Haverty <phaverty@gene.com>

See Also

Other "range summaries": boundingIndices, boundingIndices2, rangeColMeans, rangeSampleMeans

---

bounds2Rle *Convert bounding indices into a Rle*

Description

Given a matrix of first/last indices, like from boundingIndicesByChr, and values for each range, convert to a Rle. This function takes the expected length of the Rle, n, so that any portion of the full length not covered by a first/last range will be a run with the value NA. This is typical in the case where data is segmented with CBS and some of the data to be segmented is NA.

Usage

```
bounds2Rle(bounds, values, n)
```

Arguments

bounds	matrix, two columns, with first and last index, like from boundingIndicesByChr
values	ANY, some value to be associated with each range, like segmented copy number.
n	integer, the expected length of the Rle, i.e. the number of features in the genome/target ranges processed by boundingIndicesByChr.

Value

Rle

Author(s)

Peter M. Haverty

See Also

Other "segmented data": rangeSegMeanLength, rangeSegMeanLength, rangeSegMeanLength, runCBS, segPairTable, segPairTable, segPairTable, segs2Granges, segs2RangedData, segs2Rle, segs2RleDataFrame, segTable, segTable, segTable

calcGC *Calculate GC Percentage in windows*

Description

Local GC content can be used to remove GC artifacts from copynumber data see Diskin, 2008). This function will calculate GC content fraction in expanded windows around a set of ranges following example in http://www.bioconductor.org/help/course-materials/2012/useR2012/Bioconductor-tutorial.pdf. Currently all ranges are tabulated, later I may do letterFrequencyInSlidingWindow for big windows and then match to the nearest.

Usage

```
calcGC(object, bsgenome, expand = 1e+06)
```

Arguments

`object`	GenomicRanges, GenoSet, or RangedData
`bsgenome`	BSgenome, like Hsapiens from BSgenome.Hsapiens.UCSC.hg19
`expand`	scalar integer, amount to expand each range before calculating gc

Value

numeric vector, fraction of nucleotides that are G or C in expanded ranges of `object`

Examples

```
## Not run:  data(genoset)
## Not run:  library(BSgenome.Hsapiens.UCSC.hg19)
## Not run:  gc = calcGC(genoset.ds, Hsapiens)
```

chr *Look up chromosome for each feature*

Description

Chromosome name for each feature

Arguments

`object`	GRanges, RangedData or GenoSet

Details

Get chromosome name for each feature. Returns character, not the factor 'space'.

Value

character vector of chromosome positions for each feature

Author(s)

Peter Haverty

Examples

```
data(genoset)
  chr(genoset.ds)  # c("chr1","chr1","chr1","chr1","chr3","chr3","chrX","chrX","chrX","chrX")
   chr(locData(genoset.ds))  # The same
```

---

chrIndices   *Get a matrix of first and last index of features in each chromosome*

Description

Sometimes it is handy to know the first and last index for each chr. This is like chrInfo but for feature indices rather than chromosome locations. If chr is specified, the function will return a sequence of integers representing the row indices of features on that chromosome.

Arguments

`object`	GenoSet, RangedData, or GRanges
`chr`	character, specific chromosome name

Value

data.frame with "first" and "last" columns

Author(s)

Peter M. Haverty

Examples

```
data(genoset)
  chrIndices(genoset.ds)
  chrIndices(locData(genoset.ds))  # The same
```

chrInfo *Chromosome Information*

Description

Get chromosome start and stop positions

Arguments

`object`	A GenoSet object or similar

Details

Provides a matrix of start, stop and offset, in base numbers for each chromosome.

Value

list with start and stop position, by ordered chr

Author(s)

Peter Haverty

Examples

```
data(genoset)
  chrInfo(genoset.ds)
  chrInfo(locData(genoset.ds))  # The same
```

chrNames *Get list of unique chromosome names*

Description

Get list of unique chromosome names

Arguments

`object`	RangedData or GenoSet

Value

character vector with names of chromosomes

Author(s)

Peter M. Haverty

Examples

```
data(genoset)
  chrNames(genoset.ds) # c("chr1","chr3","chrX")
  chrNames(locData(genoset.ds))  # The same
  chrNames(genoset.ds) = sub("^chr","",chrNames(genoset.ds))
```

`chrOrder` *Order chromosome names in proper genome order*

Description

Chromosomes make the most sense orded by number, then by letter.

Usage

```
chrOrder(chr.names)
```

Arguments

`chr.names`	character, vector of unique chromosome names

Value

character vector of chromosome names in proper order

Author(s)

Peter M. Haverty

See Also

Other "genome ordering": `isGenomeOrder`, `isGenomeOrder`, `isGenomeOrder`, `toGenomeOrder`, `toGenomeOrder`, `toGenomeOrder`, `toGenomeOrder`

Examples

```
chrOrder(c("chr5","chrX","chr3","chr7","chrY"))  #  c("chr3","chr5","chr7","chrX","chrY")
```

`cn` *Get or Set the cn assayData slot*

Description

Get or Set the cn assayData slot

Arguments

`object`	A BAFset object

Value

matrix

Author(s)

Peter M. Haverty

`cn2lr` *Take vector or matrix of copynumber values, convert to log2ratios Utility function for converting copynumber units (2 is normal) to log2ratio units (two is normal)*

Description

Take vector or matrix of copynumber values, convert to log2ratios Utility function for converting copynumber units (2 is normal) to log2ratio units (two is normal)

Usage

```
cn2lr(x)
```

Arguments

`x`	numeric data in copynumber units

Value

data of same type as "x" transformed into log2ratio units

Author(s)

Peter M. Haverty <`phaverty@gene.com`>

See Also

lr2cn

CNSet — *Create a CNSet object*

Description

This function is the preferred method for creating a new CNSet object. Users are generally discouraged from calling "new" directly. This CNSet function enforces the requirement for a "cn" matrix. This and any other "..." arguments will become part of the assayData slot of the resulting object. "..." can be matrices or DataFrame objects (from the IRanges package). This function passes control to the "initGenoSet" method which performs argument checking including dimname matching among relevant slots and sets everything to genome order. Genome order can be disrupted by "[" or "[[" calls and will be checked by methods that require it.

Usage

```
CNSet(locData, cn = NULL, pData = NULL, annotation = "",
  universe, assayData = NULL, ...)
```

Arguments

`locData`	A GRanges or RangedData object specifying feature chromosome locations. featureNames (names or rownames) are required to match featureNames of matrices.
`cn`	numeric matrix of copy number data with rownames matching featureNames and colnames matching sampleNames
`pData`	A data frame with rownames matching all data matrices
`annotation`	character, string to specify chip/platform type
`universe`	character, string to specify genome universe for locData. Overrides any universe/genome data in locData.
`assayData`	assayData, usually an environment
`...`	More matrix or DataFrame objects to include in assayData

Details

The CNSet class is defunct. Please use GenoSet. CNSet only added the cn getter/setter functions, which are redundant with x[, , 'cn'] now.

Value

A CNSet object

Author(s)

Peter M. Haverty

CNSet-class *Class* "CNSet"

Description

A CNSet is an extension of GenoSet that requires a 'cn' assayData element.

Objects from the Class

Objects can be created by calls of the form `new("CNSet", assayData, phenoData, featureData, experimentData, ann` However, as per BioConductor standard practice the object creation function `CNSet` is recommended.

Slots

`locData`**:** Object of class `"RangedDataOrGenomicRanges"` Feature locations on the genome.

`assayData`**:** Object of class `"AssayData"` From eSet

`phenoData`**:** Object of class `"AnnotatedDataFrame"` From eSet

`featureData`**:** Object of class `"AnnotatedDataFrame"` From eSet

`experimentData`**:** Object of class `"MIAxE"` From eSet

`annotation`**:** Object of class `"character"` From eSet

`protocolData`**:** Object of class `"AnnotatedDataFrame"` From eSet

`.__classVersion__`**:** Object of class `"Versions"` From eSet

Extends

Class `"GenoSet"`, directly. Class `"eSet"`, by class "GenoSet", distance 2.

Methods

show `signature(object = "CNSet")`: ...

cn `signature(object = "CNSet")`: Getter for cn assayDataElement

cn<- `signature(object = "CNSet", value = "matrix")`: Setter for 'cn' assayDataElement

genoPlot `signature(x = "CNSet", y = "ANY")`: Plot data along the genome. Defaults to 'cn' assayDataElement

Author(s)

Peter M. Haverty <`phaverty@gene.com`>

See Also

`CNSet`, `GenoSet`, `BAFSet`

Examples

```
showClass("CNSet")
```

colMeans — *Means of columns*

Description

Calculate means of columns of a DataFrame as if it were a matrix. Allow colmeans in rangeSampleMeans for DataTable just like a real matrix. I'm sure there is much more clever way to do this using aggregate.

Arguments

`x`	DataFrame
`na.rm`	logical
`dims`	integer

Author(s)

Peter M. Haverty

Examples

```
df.ds = DataFrame( a = Rle(c(5,4,3),c(2,2,2)), b = Rle(c(3,6,9),c(1,1,4)) )
 mat.ds = matrix( c(5,5,4,4,3,3,3,6,9,9,9,9), ncol=2, dimnames=list(NULL,c("a","b")))
 ## Not run:  identical( colMeans(df.ds), colMeans(mat.ds) )
```

colnames — *Get colnames from a GenoSet*

Description

Get colnames from a GenoSet

Arguments

`object`	GenoSet

Value

character vector with names of samples

Examples

```
data(genoset)
  head(colnames(genoset.ds))
```

`featureNames` *Get rownames from RangedData, GRanges, or GenoSet*

Description

Get rownames from RangedData, GRanges, or GenoSet.

Arguments

`object`	GRanges, RangedData, or GenoSet

Value

character vector with names rows/features

Author(s)

Peter M. Haverty

Examples

```
data(genoset)
  head(rownames(locData.gr))
  head(rownames(genoset.ds))
```

`fixSegNAs` *Fix NA runs in a Rle*

Description

Fix NA runs in a Rle when the adjacent runs have equal values

Usage

```
fixSegNAs(x, max.na.run = 3)
```

Arguments

`x`	Rle to be fixed
`max.na.run`	integer, longest run of NAs that will be fixed

Value

Rle

Author(s)

Peter M. Haverty

`gcCorrect` *Correct copy number for GC content*

Description

Copy number estimates from various platforms show "Genomic Waves" (Diskin et al., Nucleic Acids Research, 2008) where copy number trends with local GC content. This function regresses copy number on GC percentage and removes the effect (returns residuals). GC content should be smoothed along the genome in wide windows >= 100kb.

Usage

```
gcCorrect(ds, gc, retain.mean = TRUE)
```

Arguments

`ds`	numeric matrix of copynumber or log2ratio values, samples in columns
`gc`	numeric vector, GC percentage for each row of ds, must not have NAs
`retain.mean`	logical, center on zero or keep same mean?

Value

numeric matrix, residuals of ds regressed on gc

Author(s)

Peter M. Haverty

Examples

```
gc = runif(n=100, min=1, max=100)
  ds = rnorm(100) + (0.1 * gc)
  gcCorrect(ds, gc)
```

`genome` *Get and set the genome universe annotation.*

Description

Genome version

Arguments

`x`	GenoSet

Details

The genome positions of the features in locData. The UCSC notation (e.g. hg18, hg19, etc.) should be used.

Value

character, e.g. hg19

Author(s)

Peter M. Haverty

Examples

```
data(genoset)
  genome(genoset.ds)
  genome(genoset.ds) = "hg19"
```

`genomeAxis` *Label axis with base pair units*

Description

Label an axis with base positions

Usage

```
genomeAxis(locs = NULL, side = 1, log = FALSE,
  do.other.side = TRUE)
```

Arguments

`locs`	RangedData to be used to draw chromosome boundaries, if necessary. Usually locData slot from a GenoSet.
`side`	integer side of plot to put axis
`log`	logical Is axis logged?
`do.other.side`	logical, label non-genome side with data values at tick marks?

Details

Label a plot with Mb, kb, bp as appropriate, using tick locations from axTicks

Value

nothing

Author(s)

Peter M. Haverty

See Also

Other "genome plots": genoPlot, genoPlot, genoPlot, genoPlot

Examples

```
data(genoset)
  genoPlot(genoPos(genoset.ds), genoset.ds[,1, "baf"])
 genomeAxis( locs=locData(genoset.ds) )  # Add chromosome names and boundaries to a plot assuming genome along x-ax
 genomeAxis( locs=locData(genoset.ds), do.other.side=FALSE ) # As above, but do not label y-axis with data values a
 genomeAxis()       # Add nucleotide position in sensible units assuming genome along x-axis
```

---

genoPlot *Plot data along the genome*

Description

Plot location data and chromosome boundaries from a GenoSet, RangedData, or GRanges object against data from a numeric or Rle. Specifying a chromosome name and optionally a 'xlim' will zoom into one chromosome region. If more than one chromosome is present, the chromosome boundaries will be marked. Alternatively, for a numeric x and a numeric or Rle y, data in y can be plotted at genome positions x. In this case, chromosome boundaries can be taken from the argument locs. If data for y-axis comes from a Rle lines are plotted representing segments. X-axis tickmarks will be labeled with genome positions in the most appropriate units.

Arguments

`x`	GenoSet (or descendant), RangedData, or GRanges
`y`	numeric or Rle
`locs`	RangedData, like locData slot of GenoSet
`chr`	Chromosome to plot, NULL by default for full genome
`add`	Add plot to existing plot
`xlab`	character, label for x-axis of plot
`ylab`	character, label for y-axis of plot
`col`	character, color to plot lines or points
`lwd`	numeric, line width for segment plots from an Rle
`pch`	character or numeric, printing character, see points
`xlim`	integer, length two, bounds for genome positions. Used in conjunction with "chr" to subset data for plotting.
`...`	Additional plotting args

Value

nothing

Methods

`signature(x = "RangedDataOrGenoSetOrGenomicRanges", y = "ANY")` Plot feature locations and data from one sample.

`signature(x = "numeric", y = "numeric")` Plot numeric location and a vector of numeric data.

`signature(x = "numeric", y = "Rle")` Plot numeric location and a vector of Rle data. Uses lines for Rle runs.

Author(s)

Peter M. Haverty

See Also

Other "genome plots": `genomeAxis`

Examples

```
data(genoset)
genoPlot( x=genoset.ds,y=genoset.ds[,1,"lrr"] )
genoPlot( genoPos(genoset.ds), genoset.ds[,1,"lrr"], locs=locData(genoset.ds) ) # The same
genoPlot( 1:10, Rle(c(rep(0,5),rep(3,4),rep(1,1))) )
```

`genoPos` *Convert chromosome positions to positions from start of genome*

Description

Get base positions of features in genome-scale units

Arguments

`object`	A GenoSet object or a RangedData object

Details

Get base positions of array features in bases counting from the start of the genome. Chromosomes are ordered numerically, when possible, then lexically.

Value

numeric position of each feature in whole genome units, in original order

Author(s)

Peter M. Haverty

Examples

```
data(genoset)
  head(genoPos(genoset.ds))
  head(genoPos(locData(genoset.ds)))  # The same
```

`GenoSet` *Create a GenoSet object*

Description

This function is the preferred method for creating a new GenoSet object. Users are generally discouraged from calling "new" directly. Any "..." arguments will become part of the assayData slot of the resulting object. "..." can be matrices or DataFrame objects (from IRanges). This function passes control to the "initGenoSet" method which performs argument checking including dimname matching among relevant slots and sets everything to genome order. Genome order can be disrupted by "[" calls and will be checked by methods that require it.

Usage

```
GenoSet(locData, pData = NULL, annotation = "", universe,
  assayData = NULL, ...)
```

Arguments

`locData`	A RangedData object specifying feature chromosome locations. Rownames are required to match featureNames.
`pData`	A data frame with rownames matching all data matrices
`annotation`	character, string to specify chip/platform type
`universe`	character, a string to specify the genome universe for locData
`assayData`	assayData, usually an environment
`...`	More matrix or DataFrame objects to include in assayData

Value

A GenoSet object

Author(s)

Peter M. Haverty

Examples

```
test.sample.names = LETTERS[11:13]
probe.names = letters[1:10]
gs = GenoSet(
  locData=GRanges(ranges=IRanges(start=1:10,width=1,names=probe.names),seqnames=c(rep("chr1",4),rep("chr3",2),
   cn=matrix(31:60,nrow=10,ncol=3,dimnames=list(probe.names,test.sample.names)),
  pData=data.frame(matrix(LETTERS[1:15],nrow=3,ncol=5,dimnames=list(test.sample.names,letters[1:5]))),
   annotation="SNP6"
)
```

`GenoSet-class` *Class* "GenoSet"

Description

`GenoSet` extends `eSet` by adding genome location information in the form of the locData slot. GenoSet uses this location information to allow quick subsetting and summarization by a set of genome locations (RangedData or GRanges). GenoSet implements and extends the RangedData/GRanges API for access to the underlying location information.

Objects from the Class

Objects can be created by calls of the form `new("GenoSet", assayData, phenoData, featureData, experimentData, a` However, as per BioConductor standard practice the object creation function `GenoSet` is recommended.

Slots

`locData`: Object of class `"RangedDataOrGenomicRanges"` Locations of features on the genome

`assayData`: Object of class `"AssayData"` From eSet

`phenoData`: Object of class `"AnnotatedDataFrame"` From eSet

`featureData`: Object of class `"AnnotatedDataFrame"` From eSet

`experimentData`: Object of class `"MIAxE"` From eSet

`annotation`: Object of class `"character"` From eSet

`protocolData`: Object of class `"AnnotatedDataFrame"` From eSet

`.__classVersion__`: Object of class `"Versions"` From eSet

Extends

Class `"eSet"`, directly.

Methods

[`signature(x = "GenoSet", i = "ANY", j = "ANY", drop = "ANY")`: ...

[`signature(x = "GenoSet", i = "character", j = "ANY", drop = "ANY")`: ...

[`signature(x = "GenoSet", i = "RangedData", j = "ANY", drop = "ANY")`: ...

[<- `signature(x = "GenoSet", i = "ANY", j = "ANY", value = "ANY")`: ...

chr `signature(object = "GenoSet")`: ...

chrNames `signature(object = "GenoSet")`: ...

elementLengths `signature(x = "GenoSet")`: ...

featureNames `signature(object = "GenoSet")`: ...

featureNames<- `signature(object = "GenoSet")`: ...

sampleNames `signature(object = "GenoSet")`: ...

dim `signature(object = "GenoSet")`: ...

genoPlot `signature(x = "GenoSet", y = "ANY")`: ...

locData `signature(object = "GenoSet")`: ...

locData<- `signature(object = "GenoSet", value = "RangedData")`: ...

names `signature(x = "GenoSet")`: ...

ranges `signature(x = "GenoSet")`: ...

show `signature(object = "GenoSet")`: ...

toGenomeOrder `signature(ds = "GenoSet")`: ...

Author(s)

Peter M. Haverty <phaverty@gene.com>

See Also

GenoSet, CNSet, BAFSet

Examples

```
showClass("GenoSet")
test.sample.names = LETTERS[11:13]
probe.names = letters[1:10]
gs = GenoSet(
  locData=GRanges(ranges=IRanges(start=1:10,width=1,names=probe.names),seqnames=c(rep("chr1",4),rep("chr3",2),
   cn=matrix(31:60,nrow=10,ncol=3,dimnames=list(probe.names,test.sample.names)),
  pData=data.frame(matrix(LETTERS[1:15],nrow=3,ncol=5,dimnames=list(test.sample.names,letters[1:5]))),
   annotation="SNP6"
)
```

genoset-datasets *Example GenoSet, BAFSet, and CNSet objects and the data to create them.*

Description

Fake LRR, BAF, pData and location data were generated and saved as fake.lrr, fake.cn, fake.baf, fake.pData and locData.rd. These were used to construct the objects genoset.ds, baf.ds, and cn.ds

Usage

```
data(genoset)
```

Format

fake.lrr A matrix with some randomly generated LRR (log2ratio copynumber) data

fake.cn A matrix with some randomly generated LRR (log2ratio copynumber) data

fake.baf A matrix with some randomly generated BAF (B-Allele Frequency) data

fake.pData A data.frame of sample annotation to go with fake.lrr and fake.baf

locData.gr A GRanges object describing the genomic locations of the probes in fake.baf and fake.lrr

genoset.ds A GenoSet object created with fake.lrr as the "foo" element, locData.rd as the locData, and fake.pData as the phenoData

Source

Fake data generated using rnorm and the like.

genoset-defunct *Defunct genoset features*

Description

The CNSet and BAFSet classes are defunct. They only really added getter/setter methods for specific assayDataElements, so they are now redundant with the preferred method of using the assayDataElement name as the third argument to bracket, e.g. `x[i, j,  "lrr"]`. Accordingly `BAFSet.to.ExpressionSets` is also defunct.

Details

Additionally, names, ranges, and space on a GenoSet are also defunct. In an effort to make a consistent API for either RangedData or GRanges in the locData slot, we recommend using `chrNames` for `names` and `chr` for `space`.

`genoset-deprecated` *Deprecated genoset features*

Description

GenoSet is moving towards using GenomicRanges instead of RangedData. We are also getting rid of dependencies on eSet for a potential switch to an underlying SummarizedExperiment.

`initGenoSet` *Create a GenoSet or derivative object*

Description

This function is the preferred method for creating a new GenoSet object. Users are generally discouraged from calling "new" directly. The "..." argument is for any number of matrices of matching size that will become part of the assayData slot of the resulting object. This function passes control to the "genoSet" object which performs argument checking including dimname matching among relevant slots and sets everything to genome order. Genome order can be disrupted by "[" calls and will be checked by methods that require it.

Usage

```
initGenoSet(type, locData, pData = NULL, annotation = "",
  universe, assayData = NULL, ...)
```

Arguments

`type`	character, the type of object (e.g. GenoSet, BAFSet, CNSet) to be created
`locData`	A GRanges or RangedData object specifying feature chromosome locations. rownames are required to match assayData.
`pData`	A data frame with rownames matching colnames of all assayDataElements
`annotation`	character, string to specify chip/platform type
`universe`	character, a string to specify the genome universe for locData, overrides universe/genome data in locData
`assayData`	assayData, usually an environment
`...`	More matrix or DataFrame objects to include in assayData

Value

A GenoSet object or derivative as specified by "type" arg

Author(s)

Peter M. Haverty

Examples

```
test.sample.names = LETTERS[11:13]
  probe.names = letters[1:10]
  gs = GenoSet(
   locData=GRanges(ranges=IRanges(start=1:10,width=1,names=probe.names),seqnames=c(rep("chr1",4),rep("chr3",2)
     cn=matrix(31:60,nrow=10,ncol=3,dimnames=list(probe.names,test.sample.names)),
   pData=data.frame(matrix(LETTERS[1:15],nrow=3,ncol=5,dimnames=list(test.sample.names,letters[1:5]))),
     annotation="SNP6"
  )
```

---

isGenomeOrder *Check if a GRanges, GenoSet or RangedData is in genome order*

Description

Checks that rows in each chr are ordered by start. If strict=TRUE, then chromosomes must be in order specified by chrOrder. isGenomeOrder for GRanges differs from order in that it orders by chromsome and start position only, rather than chromsome, strand, start, and width.

Arguments

`ds`	GenoSet, GRanges, or RangedData
`strict`	logical, should space/chromosome order be identical to that from chrOrder?

Value

logical

Author(s)

Peter M. Haverty

See Also

Other "genome ordering": `chrOrder`, `toGenomeOrder`, `toGenomeOrder`, `toGenomeOrder`, `toGenomeOrder`

Examples

```
data(genoset)
  isGenomeOrder( locData(genoset.ds) )
```

`locData` *Access the feature genome position info*

Description

The position information for each probe/feature is stored as an IRanges RangedData object. The locData functions allow this data to be accessed or re-set.

Arguments

`object`	GenoSet
`value`	RangedData describing features

Value

A GenoSet object

Methods

`signature(object = "GenoSet")` Get location data.

`signature(object = "GenoSet", value = "RangedData")` Set location data.

Author(s)

Peter M. Haverty

Examples

```
data(genoset)
rd = locData(genoset.ds)
locData(genoset.ds) = rd
```

`lr2cn` *Take vector or matrix of log2 ratios, convert to copynumber Utility function for converting log2ratio units (zero is normal) to copynumber units (two is normal)*

Description

Take vector or matrix of log2 ratios, convert to copynumber Utility function for converting log2ratio units (zero is normal) to copynumber units (two is normal)

Usage

```
lr2cn(x)
```

Arguments

`x`	numeric data in log2ratio values

Value

data of same type as "x" transformed into copynumber units

Author(s)

Peter M. Haverty <`phaverty@gene.com`>

See Also

cn2lr

`lrr` *Get lrr data*

Description

Get or Set the lrr assayData slot

Arguments

`object`	A BAFset object

Details

Get or Set the lrr assayData slot

Value

matrix

Author(s)

Peter M. Haverty

`modeCenter` *Center continuous data on mode*

Description

Copynumber data distributions are generally multi-modal. It is often assumed that the tallest peak represents "normal" and should therefore be centered on a log2ratio of zero. This function uses the density function to find the mode of the dominant peak and subtracts that value from the input data.

Usage

```
modeCenter(ds)
```

Arguments

`ds`	numeric matrix

Value

numeric matrix

Author(s)

Peter M. Haverty

Examples

```
modeCenter( matrix( rnorm(150, mean=0), ncol=3 ))
```

`pos` *Positions for features*

Description

Chromosome position of features

Arguments

`object`	GRanges, RangedData or GenoSet

Details

Get chromosome position of features/ranges. Defined as floor of mean of start and end.

Value

numeric vector of feature positions within a chromosome

Author(s)

Peter Haverty

Examples

```
data(genoset)
  pos(genoset.ds)  # 1:10
  pos(locData(genoset.ds))  # The same
```

---

rangeColMeans — *Calculate column means for multiple ranges*

Description

Essentially colMeans with a loop, all in a .Call. Designed to take a 2-column matrix of row indices, bounds, for a matrix, x, and calculate mean for each range in each column (or along a single vector). bounds matrix need not cover all rows.

Usage

```
rangeColMeans(bounds, x)
```

Arguments

`bounds`	A two column integer matrix of row indices
`x`	A numeric matrix with rows corresponding to indices in bounds.

Value

A numeric matrix or vector, matching the form of x. One row for each row in bounds, one col for each col of x and appropriate dimnames. If x is a vector, just a vector with names from the rownames of bounds.

Author(s)

Peter M. Haverty <`phaverty@gene.com`>

See Also

Other "range summaries": `boundingIndices`, `boundingIndices2`, `boundingIndicesByChr`, `rangeSampleMeans`

`rangeSampleMeans` *Average features in ranges per sample*

Description

This function takes per-feature genomic data and returns averages for each of a set of genomic ranges. The most obvious application is determining the copy number of a set of genes. The features corresponding to each gene are determined with boundingIndices such that all features with the bounds of a gene (overlaps). The features on either side of the gene unless those positions exactly match the first or last base covered by the gene. Therefore, genes falling between two features will at least cover two features. This is similar to rangeSampleMeans, but it checks the subject positions for being sorted and not being NA and also treats them as doubles, not ints. Range bounding performed by the boundingIndices function.

Usage

```
  rangeSampleMeans(query.rd, subject, assay.element)
```

Arguments

`query.rd`	RangedData object representing genomic regions (genes) to be averaged.
`subject`	A GenoSet object or derivative
`assay.element`	character, name of element in assayData to use to extract data

Value

numeric matrix of features in each range averaged by sample

Author(s)

Peter M. Haverty

See Also

Other "range summaries": `boundingIndices`, `boundingIndices2`, `boundingIndicesByChr`, `rangeColMeans`

Examples

```
data(genoset)
  my.genes = RangedData( ranges=IRanges(start=c(35e6,128e6),end=c(37e6,129e6),names=c("HER2","CMYC")), space=c("
   rangeSampleMeans( my.genes, genoset.ds, "lrr" )
```

`rangeSegMeanLength` *Get segment widths*

Description

The width of a genomic segment helps inform us about the importance of a copy number value. Focal amplifications are more interesting than broad gains, for example. Given a range of interesting regions (i.e. genes) this function determines all genomics segments covered by each gene and returns the average length of the segments covered by each gene in each sample. Often only a single segment covers a given gene in a given sample.

Arguments

`range.gr`	GRanges, genome regions of interest, usually genes
`segs`	data.frame of segments, like from segTable, or a list of these

Value

named vector of lengths, one per item in range.gr, or a range x length(segs) of these if segs is also list-like.

See Also

Other "segmented data": `bounds2Rle`, `runCBS`, `segPairTable`, `segPairTable`, `segPairTable`, `segs2Granges`, `segs2RangedData`, `segs2Rle`, `segs2RleDataFrame`, `segTable`, `segTable`, `segTable`

`readGenoSet` *Load a GenoSet from a RData file*

Description

Given a rds file or a rda file with one object (a GenoSet or related object), load it, and return.

Usage

```
readGenoSet(path)
```

Arguments

`path`	character, path to rds or rda file

Value

GenoSet or related object (only object in RData file)

Author(s)

Peter M. Haverty <phaverty@gene.com>

Examples

```
## Not run:  ds = readGenoSet("/path/to/genoset.RData")
## Not run:  ds = readGenoSet("/path/to/genoset.rda")
## Not run:  ds = readGenoSet("/path/to/genoset.rds")
```

---

runCBS *Run CBS Segmentation*

Description

Utility function to run CBS's three functions on one or more samples

Usage

```
runCBS(data, locs, return.segs = FALSE, n.cores = 1,
  smooth.region = 2, outlier.SD.scale = 4,
  smooth.SD.scale = 2, trim = 0.025, alpha = 0.001)
```

Arguments

`data`	numeric matrix with continuous data in one or more columns
`locs`	GenomicRanges, like locData slot of GenoSet
`return.segs`	logical, if true list of segment data.frames return, otherwise a DataFrame of Rle vectors. One Rle per sample.
`n.cores`	numeric, number of cores to ask mclapply to use
`smooth.region`	number of positions to left and right of individual positions to consider when smoothing single point outliers
`outlier.SD.scale`	number of SD single points must exceed smooth.region to be considered an outlier
`smooth.SD.scale`	floor used to reset single point outliers
`trim`	fraction of sample to smooth
`alpha`	pvalue cutoff for calling a breakpoint

Details

Takes care of running CBS segmentation on one or more samples. Makes appropriate input, smooths outliers, and segment

Value

data frame of segments from CBS

Author(s)

Peter M. Haverty

See Also

Other "segmented data": bounds2Rle, rangeSegMeanLength, rangeSegMeanLength, rangeSegMeanLength, segPairTable, segPairTable, segPairTable, segs2Granges, segs2RangedData, segs2Rle, segs2RleDataFrame, segTable, segTable, segTable

Examples

```
sample.names = paste("a",1:2,sep="")
    probe.names =  paste("p",1:30,sep="")
   ds = matrix(c(c(rep(5,20),rep(3,10)),c(rep(2,10),rep(7,10),rep(9,10))),ncol=2,dimnames=list(probe.names,samp
   locs = GRanges(ranges=IRanges(start=c(1:20,1:10),width=1,names=probe.names),seqnames=paste("chr",c(rep(1,20)

   seg.rle.result = DataFrame( a1 = Rle(c(rep(5,20),rep(3,10))), a2 = Rle(c(rep(2,10),rep(7,10),rep(9,10))), row.n
    seg.list.result = list(
    a1 = data.frame( ID=rep("a1",2), chrom=factor(c("chr1","chr2")), loc.start=c(1,1), loc.end=c(20,10), num.mar
    a2 = data.frame( ID=rep("a2",3), chrom=factor(c("chr1","chr1","chr2")), loc.start=c(1,11,1), loc.end=c(10,20
      )

    runCBS(ds,locs)  # Should give seg.rle.result
    runCBS(ds,locs,return.segs=TRUE) # Should give seg.list.result
```

---

segPairTable *Convert Rle objects to tables of segments*

Description

Like segTable, but for two Rle objects. Takes a pair of Rle or DataFrames with Rle columns and makes one or more data.frames with bounds of each new segment. Rle objects are broken up so that each resulting segment has one value from each Rle. For a DataFrame, the argument `stack` combines all of the individual data.frames into one large data.frame and adds a "Sample" column of sample ids.

Arguments

`x`	Rle or list/DataFrame of Rle vectors
`y`	Rle or list/DataFrame of Rle vectors
`locs`	GenomicRanges with rows corresponding to rows of df
`chr.ind`	matrix, like from chrIndices method
`start`	integer, vector of feature start positions

end	integer, vector of feature end positions
factor.chr	scalar logical, make 'chrom' column a factor?
stack	logical, rbind list of segment tables for each sample and add "Sample" column?

Details

For a Rle, the user can provide `locs` or `chr.ind`, `start` and `stop`. The latter is surprisingly much faster and this is used in the DataFrame version.

Value

one or a list of data.frames with columns chrom, loc.start, loc.end, num.mark, seg.mean

Author(s)

Peter M. Haverty

See Also

Other "segmented data": `bounds2Rle`, `rangeSegMeanLength`, `rangeSegMeanLength`, `rangeSegMeanLength`, `runCBS`, `segs2Granges`, `segs2RangedData`, `segs2Rle`, `segs2RleDataFrame`, `segTable`, `segTable`, `segTable`

Examples

```
cn = Rle(c(3,4,5,6),rep(3,4))
  loh = Rle(c(2,4,6,8,10,12),rep(2,6))
  start = c(9:11,4:9,15:17)
  end = start
 locs = GRanges(IRanges(start=start,end=end),seqnames=c(rep("chr1",3),rep("chr2",6),rep("chr3",3)))
  segPairTable(cn,loh,locs)
```

`segs2Granges` *GRanges from segment table*

Description

GenoSet contains a number of functions that work on segments. Many work on a data.frame of segments, like segTable and runCBS. This function converts one of these tables in a GRanges. The three columns specifying the ranges become the GRanges and all other columns go into the 'mcols' portion of the GRanges object.

Usage

```
segs2Granges(segs)
```

Arguments

`segs`	data.frame with loc.start, loc.end, and chrom columns, like from segTable or runCBS

Value

GRanges

See Also

Other "segmented data": `bounds2Rle`, `rangeSegMeanLength`, `rangeSegMeanLength`, `rangeSegMeanLength`, `runCBS`, `segPairTable`, `segPairTable`, `segPairTable`, `segs2RangedData`, `segs2Rle`, `segs2RleDataFrame`, `segTable`, `segTable`, `segTable`

`segs2RangedData` *Make a RangedData from segments*

Description

Starting from a data.frame of segments, like from CBS and segTable, organize as a RangedData. Label data "score", so it can easily be made into various genome browser formats using rtracklayer.

Usage

```
segs2RangedData(segs)
```

Arguments

`segs`	data.frame, like from segment in DNAcopy or segTable

Value

RangedData

Author(s)

Peter M. Haverty <`phaverty@gene.com`>

See Also

Other "segmented data": `bounds2Rle`, `rangeSegMeanLength`, `rangeSegMeanLength`, `rangeSegMeanLength`, `runCBS`, `segPairTable`, `segPairTable`, `segPairTable`, `segs2Granges`, `segs2Rle`, `segs2RleDataFrame`, `segTable`, `segTable`, `segTable`

Other segments: `segs2RleDataFrame`

segs2Rle — *Make Rle from segments for one sample*

Description

Take output of CBS, make Rle representing all features in 'locs' ranges. CBS output contains run length and run values for genomic segmetns, which could very directly be converted into a Rle. However, as NA values are often removed, especially for mBAF data, these run lengths do not necessarily cover all features in every sample. Using the start and top positions of each segment and the location of each feature, we can make a Rle that represents all features.

Usage

```
segs2Rle(segs, locs)
```

Arguments

`segs`	data.frame of segments, formatted as output of segment function from DNAcopy package
`locs`	GenomicRanges, like locData slot of a GenoSet

Value

Rle with run lengths and run values covering all features in the data set.

Author(s)

Peter M. Haverty <phaverty@gene.com>

See Also

Other "segmented data": `bounds2Rle`, `rangeSegMeanLength`, `rangeSegMeanLength`, `rangeSegMeanLength`, `runCBS`, `segPairTable`, `segPairTable`, `segPairTable`, `segs2Granges`, `segs2RangedData`, `segs2RleDataFrame`, `segTable`, `segTable`, `segTable`

Examples

```
data(genoset)
  segs = runCBS( genoset.ds[, , "lrr"], locData(genoset.ds), return.segs=TRUE )
 segs2Rle( segs[[1]], locData(genoset.ds) )  # Take a data.frame of segments, say from DNAcopys segment function, a
```

segs2RleDataFrame *CBS segments to probe matrix*

Description

Given segments, make a DataFrame of Rle objects for each sample

Usage

```
segs2RleDataFrame(seg.list, locs)
```

Arguments

`seg.list`	list, list of data frames, one per sample, each is result from CBS
`locs`	locData from a GenoSet object

Details

Take table of segments from CBS, convert DataTable of Rle objects for each sample.

Value

DataFrame of Rle objects with nrows same as locs and one column for each sample

Author(s)

Peter Haverty

See Also

Other "segmented data": `bounds2Rle`, `rangeSegMeanLength`, `rangeSegMeanLength`, `rangeSegMeanLength`, `runCBS`, `segPairTable`, `segPairTable`, `segPairTable`, `segs2Granges`, `segs2RangedData`, `segs2Rle`, `segTable`, `segTable`, `segTable`

Other segments: `segs2RangedData`

Examples

```
data(genoset)
  seg.list = runCBS( genoset.ds[, , "lrr"], locData(genoset.ds), return.segs=TRUE )
 segs2RleDataFrame( seg.list, locData(genoset.ds) )  # Loop segs2Rle on list of data.frames in seg.list
```

`segTable` *Convert Rle objects to tables of segments*

Description

Like the inverse of segs2Rle and segs2RleDataFrame. Takes a Rle or a DataFrame with Rle columns and the locData both from a GenoSet object and makes a list of data.frames each like the result of CBS's segment. Note the loc.start and loc.stop will correspond exactly to probe locations in locData and the input to segs2RleDataFrame are not necessarily so. For a DataFrame, the argument `stack` combines all of the individual data.frames into one large data.frame and adds a "Sample" column of sample ids.

Arguments

`object`	Rle or list/DataFrame of Rle vectors
`locs`	GenomicRanges with rows corresponding to rows of df
`chr.ind`	matrix, like from chrIndices method
`start`	integer, vector of feature start positions
`end`	integer, vector of feature end positions
`factor.chr`	scalar logical, make 'chrom' column a factor?
`stack`	logical, rbind list of segment tables for each sample and add "Sample" column?

Details

For a Rle, the user can provide `locs` or `chr.ind`, `start` and `stop`. The latter is surprisingly much faster and this is used in the DataFrame version.

Value

one or a list of data.frames with columns chrom, loc.start, loc.end, num.mark, seg.mean

Author(s)

Peter M. Haverty

See Also

Other "segmented data": `bounds2Rle`, `rangeSegMeanLength`, `rangeSegMeanLength`, `rangeSegMeanLength`, `runCBS`, `segPairTable`, `segPairTable`, `segPairTable`, `segs2Granges`, `segs2RangedData`, `segs2Rle`, `segs2RleDataFrame`

Examples

```
data(genoset)
  seg.list = runCBS( genoset.ds[, , "lrr"], locData(genoset.ds), return.segs=TRUE )
 df = segs2RleDataFrame( seg.list, locData(genoset.ds) )  # Loop segs2Rle on list of data.frames in seg.list
  assayDataElement( genoset.ds, "lrr.segs" ) = df
  segTable( df, locData(genoset.ds) )
  segTable( genoset.ds[ , , "lrr.segs"], locData(genoset.ds) )
  segTable( genoset.ds[ , 1, "lrr.segs"], locData(genoset.ds), colnames(genoset.ds)[1] )
```

---

show *Show a GenoSet*

Description

Prints a description of a GenoSet object.

---

start *Get start of location for each feature*

Description

Get start of location for each feature

Get end of location for each feature

Get width of location for each feature

Get names of data matrices. For the time being, this is `assayDataElementNames`. This function used to do `chrNames`.

Get elementLengths from locData slot

Arguments

`x`	GenoSet
`x`	GenoSet
`x`	GenoSet
`x`	GenoSet
`x`	GenoSet
`x`	GenoSet
`i`	character, RangedData, logical, integer
`j`	character, RangedData, logical, integer
`k`	character or integer
`drop`	logical drop levels of space factor?
`...`	additional subsetting args

Details

Get elementLengths from locData slot

Value

integer

integer

integer

character

character

Author(s)

Peter M. Haverty

Peter M. Haverty

Peter M. Haverty

Peter Haverty

Peter Haverty

Examples

```
data(genoset)
  genoset.ds[1:5,2:3]  # first five probes and samples 2 and 3
  genoset.ds[ , "K"]  # Sample called K
 rd = RangedData(ranges=IRanges(start=seq(from=15e6,by=1e6,length=7),width=1),names=letters[8:14],space=rep("ch
 genoset.ds[ rd, "K" ]  # sample K and probes overlapping those in rd, which overlap specifed ranges on chr17
```

`subsetAssayData` *Subset assayData*

Description

Subset or re-order assayData

Usage

```
subsetAssayData(orig, i, j, ..., drop = FALSE)
```

Arguments

`orig`	assayData environment
`i`	row indices
`j`	col indices
`...`	Additional args to give to subset operator
`drop`	logical, drop dimensions when subsetting with single value?

Details

Subset or re-order assayData locked environment, environment, or list. Shamelessly stolen from "[" method in Biobase version 2.8 along with guts of assayDataStorageMode()

Value

assayData data structure

Author(s)

Peter M. Haverty

Examples

```
data(genoset)
  ad = assayData(genoset.ds)
  small.ad = subsetAssayData(ad,1:5,2:3)
```

`toGenomeOrder` *Set a GRanges, GenoSet, or RangedData to genome order*

Description

Returns a re-ordered object sorted by chromosome and start position. If strict=TRUE, then chromosomes must be in order specified by chrOrder. If ds is already ordered, no re-ordering is done. Therefore, checking order with isGenomeOrder, is unnecessary if order will be corrected if isGenomeOrder is FALSE.

Arguments

`ds`	GenoSet, GRanges, or RangedData
`strict`	logical, should chromosomes be in order specified by chrOrder?

Details

toGenomeOrder for GRanges differs from sort in that it orders by chromsome and start position only, rather than chromsome, strand, start, and width.

Value

re-ordered ds

Author(s)

Peter M. Haverty

See Also

Other "genome ordering": chrOrder, isGenomeOrder, isGenomeOrder, isGenomeOrder

Examples

```
data(genoset)
  toGenomeOrder( genoset.ds, strict=TRUE )
  toGenomeOrder( genoset.ds, strict=FALSE )
  toGenomeOrder( locData(genoset.ds) )
```

Index