genoset

October 25, 2011

`BAFSet` *Create a BAFSet object...*

Description

Create a BAFSet object

Usage

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

Arguments

`locData`	A RangedData object specifying feature chromosome locations. Rownames are required to match featureNames.
`lrr`	numeric matrix of copy number data with rownames matching sampleNames and colnames matching sampleNames
`baf`	numeric matrix of B-Allele Frequency data with rownames matching sampleNames 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
`...`	More matrix or DataFrame objects to include in assayData slot

Details

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.

Value

A BAFSet object

Author(s)

Peter M. Haverty

See Also

bafset-class, genoset-class

Examples

```
test.sample.names = LETTERS[11:13]
probe.names = letters[1:10]
locData.rd = RangedData(ranges=IRanges(start=c(1,4,3,2,5:10),width=1,names=probe.names),s
bs = BAFSet(
locData=locData.rd,
lrr=matrix(1:30,nrow=10,ncol=3,dimnames=list(probe.names,test.sample.names)),
baf=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,lette
annotation="SNP6"
)
```

BAFSet.to.ExpressionSets

Make a pair of ExpressionSets from a BAFSet...

Description

Make a pair of ExpressionSets from a BAFSet

Usage

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

Arguments

`bs`	A BAFset object

Details

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").

Value

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

Author(s)

Peter M. Haverty

Examples

```
data(genoset)
eset.list = BAFSet.to.ExpressionSets(baf.ds)
```

`CNSet` *Create a CNSet object...*

Description

Create a CNSet object

Usage

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

Arguments

`locData`	A RangedData object specifying feature chromosome locations. Rownames are required to match featureNames.
`cn`	numeric matrix of copy number data with rownames matching sampleNames 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
`...`	More matrix or DataFrame objects to include in assayData

Details

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.

Value

A CNSet object

Author(s)

Peter M. Haverty

Examples

```
test.sample.names = LETTERS[11:13]
probe.names = letters[1:10]
joe = CNSet(
locData=RangedData(ranges=IRanges(start=1:10,width=1,names=probe.names),space=c(rep("chr1
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,lette
annotation="SNP6"
)
```

`GenoSet` *Create a GenoSet object...*

Description

Create a GenoSet object

Usage

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

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

`...` More matrix or DataFrame objects to include in assayData

Details

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 "[" or "[[" calls and will be checked by methods that require it.

Value

A GenoSet object

Author(s)

Peter M. Haverty

Examples

```
test.sample.names = LETTERS[11:13]
probe.names = letters[1:10]
gs = GenoSet(
locData=RangedData(ranges=IRanges(start=1:10,width=1,names=probe.names),space=c(rep("chr1
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,lette
annotation="SNP6"
)
```

`baf` *Get or Set the baf assayData slot...*

Description

Get or Set the baf assayData slot

Arguments

`object`	A BAFset object

Details

`baf-methods`: Get or Set the baf assayData slot

Value

`baf-methods`: matrix

Author(s)

Peter M. Haverty

Examples

```
data(genoset)
baf(baf.ds)  # Returns assayDataElement called "baf"
baf(baf.ds) <- baf2mbaf( baf(baf.ds) )
```

`baf2mbaf` *Calculate mBAF from BAF...*

Description

Calculate mBAF from BAF

Usage

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

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.

Details

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'.

Value

numeric matix of mBAF values

Author(s)

Peter M. Haverty

Examples

```
data(genoset)
mbaf = baf2mbaf( baf(baf.ds), hom.cutoff=0.9 )
calls = matrix(sample(c("AT","AA","CG","GC","AT","GG"),(nrow(baf.ds) * 2),replace=TRUE),r
mbaf = baf2mbaf( baf(baf.ds), hom.cutoff=0.9, calls = calls, call.pairs = list(K="L",L="L
assayDataElement(baf.ds,"mbaf") = baf2mbaf( baf(baf.ds), hom.cutoff=0.9 ) # Put mbaf back
```

`bafset-class` *BAFSet class*

Description

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

Extends

`GenoSet`

Author(s)

Peter M. Haverty

See Also

bafset-class, cnset-class

Examples

```
## Creating a BAFSet
test.sample.names = LETTERS[11:13]
probe.names = letters[1:10]
locData.rd = RangedData(ranges=IRanges(start=c(1,4,3,2,5:10),width=1,names=probe.names),s
bs = BAFSet(
  locData=locData.rd,
  lrr=matrix(1:30,nrow=10,ncol=3,dimnames=list(probe.names,test.sample.names)),
  baf=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,let
  annotation="SNP6"
)
```

`boundingIndices` *Find indices of features bounding a set of chromsome ranges/genes...*

Description

Find indices of features bounding a set of chromsome ranges/genes

Usage

```
boundingIndices(starts, stops, positions, valid.indices=TRUE, initial.bounds, al
```

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

`initial.bounds` vector of length 2, first and last index of positions to use in search. For example bounds of a chromosome in whole genome base positions

`all.indices` logical, return a list containing full sequence of indices for each query

Details

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.

This function uses the trick 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 intervalBound, which uses findInterval: boundingIndices does not check for NAs or unsorted data in the subject positions. Also, the subject positions are kept as integer, where intervalBound (and findInterval) convert them to doubles. These three once-per-call differences account for much of the speed improvement in boundingIndices. These three differences are meant for position info coming from GenoSet objects and intervalBound is safer for general use.

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

intervalBound

Examples

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

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

Description

Find indices of features bounding a set of chromsome ranges/genes

Usage

```
boundingIndices2(starts, stops, positions, initial.bounds)
```

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
`initial.bounds`	numeric, length 2, first and last index of portion of positions to do search in (e.g. one chr in a genome)

Details

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.

Value

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

Author(s)

Peter M. Haverty

Examples

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

`chr-methods` *Look up chromosome for each feature*

Description

Chromsome name for each feature

Arguments

`object`	RangedData or GenoSet

Details

`chr-methods`: Get chromosome name for each feature. Returns character, not the factor 'space'.

Value

`chr-methods`: character vector of chromosome positions for each feature

Author(s)

Peter Haverty

Examples

```
test.sample.names = LETTERS[11:13]
probe.names = letters[1:10]
gs = GenoSet(
locData=RangedData(ranges=IRanges(start=1:10,width=1,names=probe.names),space=c(rep("chr1
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,lette
annotation="SNP6"
)
chr(gs)  # c("chr1","chr1","chr1","chr1","chr3","chr3","chrX","chrX","chrX","chrX")
chr(locData(gs))  # The same
```

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

Description

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

Arguments

`object`	GenoSet or RangedData

Details

`chrIndices-methods`: 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.

Value

`chrIndices-methods`: 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

`chrInfo-methods`: Provides a matrix of start, stop and offset, in base numbers for each chromosome.

Value

`chrInfo-methods`: 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
```

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

Description

Order chromosome names in proper genome order

Usage

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

Arguments

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

Details

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

Value

character vector of chromosome names in proper order

Author(s)

Peter M. Haverty

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

Details

`cn-methods`: Get or Set the cn assayData slot

Value

`cn-methods`: matrix

Author(s)

Peter M. Haverty

Examples

```
data(genoset)
cn(cn.ds)  # Returns assayDataElement called "cn"
cn(cn.ds) <- cn(cn.ds) + 5
```

`cnset-class` *CNSet class*

Description

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

Extends

`GenoSet`

Author(s)

Peter M. Haverty

See Also

bafset-class, cnset-class

Examples

```
test.sample.names = LETTERS[11:13]
probe.names = letters[1:10]
cn.ds = CNSet(
   locData=RangedData(ranges=IRanges(start=1:10,width=1,names=probe.names),space=c(rep("c
   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,le
   annotation="SNP6"
   )
```

`colMeans` *Means of columns...*

Description

Means of columns

Arguments

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

Details

`colMeans-methods`: Get means of columns of a DataFrame as if it were a matrix

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")))
identical( colMeans(df.ds), colMeans(mat.ds) )
```

`gcCorrect` *cgCorrect*

Description

Correct copy number for GC content

Usage

```
gcCorrect(ds, gc)
```

Arguments

`ds` numeric matrix of copynumber or log2ratio values, samples in columns

`gc` numeric vector, GC percentage for each row of ds

Details

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.

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)
```

`genoPlot` *genoPlot,-method*

Description

Plot data along the genome

Arguments

`sample`	A index or sampleName to plot
`element`	character, name of element in assayData to plot
`x`	GenoSet (or descendant) or numeric with chromosome or genome positions
`y`	numeric or Rle, values to be used for y-dimension, run start and stop indices or numeric with all values mapped to values in x for x-dimension or index of sample to be plotted if x is a GenoSet.
`element`	character, when x is a GenoSet, the name of the assayDataElement to plot from.
`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 charactater, see points
`...`	Additional plotting args

Details

`genoPlot-methods`: For a GenoSet object, data for a specified sample in a specified assayDataElement can be plotted along the genome. One chromosome can be specified if desired. 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 y. In this case, chromosome boundaries can be taken from the argument locs. If data for y-axis comes from a Rle, either specified directly or coming from the specified assayData element and sample, lines are plotted representing segments.

Value

`genoPlot-methods`: nothing

Author(s)

Peter M. Haverty

Examples

```
data(genoset)
genoPlot( baf.ds,1,element="lrr")
genoPlot( genoPos(baf.ds), assayDataElement(baf.ds,"lrr")[,1], locs=locData(baf.ds) ) # T
genoPlot( 1:10, Rle(c(rep(0,5),rep(3,4),rep(1,1))) )
```

genoPos-methods *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

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

Value

`genoPos-methods`: numeric position of each feature in whole genome units, in original order

Author(s)

Peter M. Haverty

Examples

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

genomeAxis *Label axis with base pair units*

Description

Label an axis with base positions

Usage

```
genomeAxis(locs, 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

Examples

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

---

genomeOrder *Get indices to set a RangedData or GenoSet to genome order...*

Description

Get indices to set a RangedData or GenoSet to genome order

Usage

```
genomeOrder(ds, strict=FALSE)
```

Arguments

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

Details

Returns a vector of idices to use in re-ordering a RangedData or GenoSet to genome order. If strict=TRUE, then chromsomes must be in order specified by chrOrder.

Value

numeric vector of indices for re-ordering

Author(s)

Peter M. Haverty

Examples

```
data(genoset)
genomeOrder( baf.ds )
genomeOrder( baf.ds, strict=TRUE )
```

genoset-class *GenoSet class*

Description

The genoset package offers an extension of the BioConductor eSet object for genome arrays. The package offers three classes. The first class is the `GenoSet` class which can hold an arbitrary number of equal-sized matrices in its assayData slot. The principal addition of the GenoSet class is a `locData` slot that holds a RangedData object from the IRanges package. The locData slot allows for quick subsetting by genome position.

Two classes extend GenoSet: CNSet and BAFSet. CNSet is the basic copy number object. It keeps its data in the `cn` slot, similar to the `exprs` slot of the ExpressionSet. BAFSet is intended to store `LRR` or Log-R Ratio and `BAF` or B-Allele Frequency data for SNP arrays. LRR and BAF come from the terms coined by Illumina. LRR is copynumber data processed on a per-snp basis to remove some variability using the expected log-ratio of normal samples with the same genotype. BAF represents the fraction of signal coming from the “B” allele, relative to the “A” allele, where A and B are arbitrarily assigned. BAF has the expected value of 0 or 1 for HOM alleles and 0.5 for HET alelles. Deviation from these expected values can be interpreted as Allelic Imbalance, which is a sign of gain, loss, or copy-neutral LOH.

Slots

`locData`: (`RangedData`) Contains a RangedData that holds probe locations

Extends

`eSet`

Author(s)

Peter M. Haverty

See Also

bafset-class, cnset-class

Examples

```
## Creating a GenoSet
test.sample.names = LETTERS[11:13]
probe.names = letters[1:10]
gs = GenoSet(
   locData=RangedData(ranges=IRanges(start=1:10,width=1,names=probe.names),space=c(rep("c
   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,le
   annotation="SNP6"
)
```

`genoset` *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.baf, fake.pData and locData.rd. These were used to construct the objects genoset.ds, baf.ds, and cn.ds

Usage

```
genoset
```

Format

A vector containing 141 observations.

Source

Fake data generated using rnorm and the like.

`genoset-methods` *Get space factor for GenoSet...*

Description

Get space factor for GenoSet

Usage

```
## S4 method for signature 'GenoSet,ANY,ANY'
x[i, j, ..., drop=FALSE]
## S4 method for signature 'GenoSet,character,ANY'
x[i, j, ..., drop=FALSE]
## S4 method for signature 'GenoSet,RangedData,ANY'
x[i, j, ..., drop=FALSE]
## S4 method for signature 'GenoSet,RangesList,ANY'
x[i, j, ..., drop=FALSE]
## S4 method for signature 'GenoSet,character'
x[[i, ..., drop=FALSE]]
```

Arguments

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

Details

`space,-method`: locData slot holds a RangedData, which keeps the chromsome of each feature in a factor names 'space'.

`start,-method`: locData slot holds a RangedData.

`end,-method`: locData slot holds a RangedData.

`names,-method`: Get chromosome names, which are the names of the locData slot.

`ranges,-method`: Get ranges from locData slot

`elementLengths,-method`: Get elementLengths from locData slot

Value

`space,-method`: factor

`start,-method`: integer

`end,-method`: integer

`names,-method`: character

`ranges,-method`: character

`elementLengths,-method`: character

Author(s)

Peter M. Haverty

Examples

```
data(genoset)
space(genoset.ds)
start(genoset.ds)
end(genoset.ds)
names(genoset.ds)
ranges(genoset.ds) # Returns a RangesList
elementLengths(genoset.ds) # Returns the number of probes per chromosomedata(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=letter
genoset.ds[ rd, "K" ]  # sample K and probes overlapping those in rd, which overlap speci
genoset.ds[[ "chr8" ]]    # All samples and probes for chromosome 8
```

initGenoSet *Create a GenoSet or derivative object...*

Description

Create a GenoSet or derivative object

Usage

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

Arguments

`type` character, the type of object (e.g. GenoSet, BAFSet, CNSet) to be created

`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

`...` More matrix or DataFrame objects to include in assayData

Details

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 "[" or "[[" calls and will be checked by methods that require it.

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=RangedData(ranges=IRanges(start=1:10,width=1,names=probe.names),space=c(rep("chr1
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,lette
annotation="SNP6"
)
```

`isGenomeOrder` *Check if a RangedData or GenoSet is in genome order...*

Description

Check if a RangedData or GenoSet is in genome order

Usage

```
isGenomeOrder(ds, strict=FALSE)
```

Arguments

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

Details

Checks that rows in each chr are ordered by start. If strict=TRUE, then chromsomes must be in order specified by chrOrder.

Value

logical

Author(s)

Peter M. Haverty

Examples

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

`locData` *Get and set probe set info*

Description

Access the feature genome position info

Arguments

`object`	GenoSet
`object`	A GenoSet object
`value`	RangedData describing features

Details

`locData-methods`: 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.

`locData<-,-method`: Set locData

Value

`locData<-,-method`: A GenoSet object

Author(s)

Peter M. Haverty

Examples

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

`lrr` *Get or Set the lrr assayData slot...*

Description

Get or Set the lrr assayData slot

Arguments

`object`	A BAFset object

Details

`lrr-methods`: Get or Set the lrr assayData slot

Value

`lrr-methods`: matrix

Author(s)

Peter M. Haverty

Examples

```
data(genoset)
lrr(baf.ds)  # Returns assayDataElement called "lrr"
lrr(baf.ds) <- lrr(baf.ds) + 0.1
```

`modeCenter` *Center continuous data on mode...*

Description

Center continuous data on mode

Usage

```
modeCenter(ds)
```

Arguments

`ds`	numeric matrix

Details

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.

Value

numeric matrix

Author(s)

Peter M. Haverty

Examples

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

orderedChrs *Get chromosome names in genome order..*

Description

Get chromosome names in genome order

Arguments

`object`	GenoSet or RangedData

Details

`orderedChrs-methods`: Get chromosome names from locData data in a GenoSet. Order numerically, for numeric chromosomes, then lexically for the rest.

Value

`orderedChrs-methods`: character vector with chrs in genome order

Author(s)

Peter M. Haverty

Examples

```
test.sample.names = LETTERS[11:13]
probe.names = letters[1:10]
gs = GenoSet(
locData=RangedData(ranges=IRanges(start=1:10,width=1,names=probe.names),space=c(rep("chr1
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,lette
annotation="SNP6"
)
orderedChrs(gs) # c("chr1","chr3","chrX")
orderedChrs(locData(gs))  # The same
```

pos *Positions for features*

Description

Chromosome position of features

Arguments

`object`	RangedData or GenoSet

Details

`pos-methods`: Get chromsome position of features/ranges. Defined as floor of mean of start and end.

Value

`pos-methods`: numeric vector of feature positions within a chromosome

Author(s)

Peter Haverty

Examples

```
test.sample.names = LETTERS[11:13]
probe.names = letters[1:10]
gs = GenoSet(
locData=RangedData(ranges=IRanges(start=1:10,width=1,names=probe.names),space=c(rep("chr1
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,lette
annotation="SNP6"
)
pos(gs)  # 1:10
pos(locData(gs))  # The same
```

rangeSampleMeans *Average features in ranges per sample...*

Description

Average features in ranges per sample

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

Details

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.

Value

numeric matrix of features in each range averaged by sample

Author(s)

Peter M. Haverty

See Also

boundingIndices intervalBound

Examples

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

`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=getOption("cores"), smooth.region=
```

Arguments

`data` numeric matrix with continuous data in one or more columns

`locs` RangeData, 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 multicore 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

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

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=li
locs = RangedData(ranges=IRanges(start=c(1:20,1:10),width=1,names=probe.names),space=past

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

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

segTable *Take a DataFrame of Rle vectors and make a list of data...*

Description

Take a DataFrame of Rle vectors and make a list of data.frames

Usage

```
segTable(df, locs)
```

Arguments

`df`	list or DataFrame of Rle vectors
`locs`	RangedData with rows corresponding to rows of df

Details

Like the inverse of segs2RleDataFrame. Take a DataFrame with Rle columns and the locData RangedData both from a GenoSet object and make 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.

Value

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

Author(s)

Peter M. Haverty

Examples

```
data(genoset)
seg.list = runCBS( lrr(baf.ds), locData(baf.ds), return.segs=TRUE )
df = segs2RleDataFrame( seg.list, locData(baf.ds) )  # Loop segs2Rle on list of data.fram
assayDataElement( baf.ds, "lrr.segs" ) = df
segTable( df, locData(baf.ds) )
segTable( assayDataElement(baf.ds,"lrr.segs"), locData(baf.ds) )
```

segs2Rle *Make Rle from segments for one sample...*

Description

Make Rle from segments for one sample

Usage

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

Arguments

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

Details

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.

Value

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

Author(s)

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

Examples

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

`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

Examples

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

`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)
```

uniqueChrs *Get list of unique chromosome names...*

Description

Get list of unique chromosome names

Arguments

`object`	RangedData or GenoSet

Details

`uniqueChrs-methods`: Get list of unique chromosome names. A synonym for names().

Value

`uniqueChrs-methods`: character vector with names of chromosomes

Author(s)

Peter M. Haverty

Examples

```
test.sample.names = LETTERS[11:13]
probe.names = letters[1:10]
gs = GenoSet(
locData=RangedData(ranges=IRanges(start=1:10,width=1,names=probe.names),space=c(rep("chr1
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,lette
annotation="SNP6"
)
uniqueChrs(gs) # c("chr1","chr3","chrX")
uniqueChrs(locData(gs))  # The same
```

universe *Get and set the genome universe annotation.*

Description

Genome universe for locData

Arguments

`x`	GenoSet
`value`	character, new universe string, e.g. hg19

Details

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

`universe<-,-method`: Set genome universe

Value

`universe,-method`: character, e.g. hg19

`universe<-,-method`: A GenoSet object

Author(s)

Peter M. Haverty

Examples

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

Index

∗Topic **classes**
bafset-class, 6
cnset-class, 13
genoset-class, 18
∗Topic **datasets**
genoset, 19
∗Topic **methods**
bafset-class, 6
cnset-class, 13
genoset-class, 18
[*(genoset-methods)*, 19
[,GenoSet,ANY,ANY-method *(genoset-methods)*, 19
[,GenoSet,ANY-method *(genoset-class)*, 18
[,GenoSet,RangedData,ANY-method *(genoset-methods)*, 19
[,GenoSet,RangedData-method *(genoset-class)*, 18
[,GenoSet,RangesList,ANY-method *(genoset-methods)*, 19
[,GenoSet,RangesList-method *(genoset-class)*, 18
[,GenoSet,character,ANY-method *(genoset-methods)*, 19
[,GenoSet,character-method *(genoset-class)*, 18
[[*(genoset-methods)*, 19
[[,GenoSet,character-method *(genoset-methods)*, 19

baf, 5
baf,-method *(baf)*, 5
baf,BAFSet-method *(bafset-class)*, 6
baf-methods *(baf)*, 5
baf.ds *(genoset)*, 19
baf2mbaf, 5
baf<- *(baf)*, 5
baf<-,-method *(baf)*, 5
baf<-,BAFSet,matrix-method *(bafset-class)*, 6
baf<-methods *(baf)*, 5
BAFSet, 1
BAFSet-class *(bafset-class)*, 6
bafset-class, 6, *6*, *13*, *18*
BAFSet.to.ExpressionSets, 2
boundingIndices, 7
boundingIndices2, 8

chr *(chr-methods)*, 9
chr,-method *(chr-methods)*, 9
chr,GenoSet-method *(genoset-class)*, 18
chr,RangedData-method *(genoset-class)*, 18
chr-methods, 9
chrIndices *(chrIndices-methods)*, 10
chrIndices,-method *(chrIndices-methods)*, 10
chrIndices,RangedDataOrGenoSet-method *(genoset-class)*, 18
chrIndices-methods, 10
chrInfo, 11
chrInfo,-method *(chrInfo)*, 11
chrInfo,RangedDataOrGenoSet-method *(genoset-class)*, 18
chrInfo-methods *(chrInfo)*, 11
chrOrder, 11
class:BAFSet *(bafset-class)*, 6
class:CNSet *(cnset-class)*, 13
class:GenoSet *(genoset-class)*, 18
cn, 12
cn,-method *(cn)*, 12
cn,CNSet-method *(cnset-class)*, 13
cn-methods *(cn)*, 12
cn.ds *(genoset)*, 19
cn<- *(cn)*, 12
cn<-,-method *(cn)*, 12
cn<-,CNSet,matrix-method *(cnset-class)*, 13
cn<-methods *(cn)*, 12
CNSet, 3
CNSet-class *(cnset-class)*, 13
cnset-class, *6*, 13, *13*, *18*
coerce,CNSet,BAFSet-method *(bafset-class)*, 6

coerce,ExpressionSet,BAFSet-method *(bafset-class)*, 6
coerce,ExpressionSet,CNSet-method *(cnset-class)*, 13
colMeans, 13
colMeans,-method *(colMeans)*, 13
colMeans,DataFrame-method *(genoset-class)*, 18
colMeans-methods *(colMeans)*, 13

elementLengths *(genoset-methods)*, 19
elementLengths,-method *(genoset-methods)*, 19
elementLengths,GenoSet-method *(genoset-class)*, 18
end *(genoset-methods)*, 19
end,-method *(genoset-methods)*, 19
end,GenoSet-method *(genoset-class)*, 18
eSet, *18*

fake.baf *(genoset)*, 19
fake.lrr *(genoset)*, 19
fake.pData *(genoset)*, 19
featureNames<-,GenoSet-method *(genoset-class)*, 18

gcCorrect, 14
genomeAxis, 16
genomeOrder, 17
genoPlot, 15
genoPlot,-method *(genoPlot)*, 15
genoPlot,BAFSet,ANY-method *(bafset-class)*, 6
genoPlot,CNSet,ANY-method *(cnset-class)*, 13
genoPlot,GenoSet,ANY-method *(genoset-class)*, 18
genoPlot,numeric,numeric-method *(genoset-class)*, 18
genoPlot,numeric,Rle-method *(genoset-class)*, 18
genoPlot-methods *(genoPlot)*, 15
genoPos *(genoPos-methods)*, 16
genoPos,-method *(genoPos-methods)*, 16
genoPos,RangedDataOrGenoSet-method *(genoset-class)*, 18
genoPos-methods, 16
GenoSet, 4, *6*, *13*
genoset, 19
GenoSet-class *(genoset-class)*, 18
genoset-class, 18
genoset-methods, 19

initGenoSet, 20
isGenomeOrder, 21

locData, 22
locData,-method *(locData)*, 22
locData,GenoSet-method *(genoset-class)*, 18
locData-methods *(locData)*, 22
locData.rd *(genoset)*, 19
locData<- *(locData)*, 22
locData<-,-method *(locData)*, 22
locData<-,GenoSet,RangedData-method *(genoset-class)*, 18
lrr, 23
lrr,-method *(lrr)*, 23
lrr,BAFSet-method *(bafset-class)*, 6
lrr-methods *(lrr)*, 23
lrr<- *(lrr)*, 23
lrr<-,-method *(lrr)*, 23
lrr<-,BAFSet,matrix-method *(bafset-class)*, 6
lrr<-methods *(lrr)*, 23

modeCenter, 23

names *(genoset-methods)*, 19
names,-method *(genoset-methods)*, 19
names,GenoSet-method *(genoset-class)*, 18

orderedChrs, 24
orderedChrs,-method *(orderedChrs)*, 24
orderedChrs,RangedDataOrGenoSet-method *(genoset-class)*, 18
orderedChrs-methods *(orderedChrs)*, 24

pos, 25
pos,-method *(pos)*, 25
pos,GenoSet-method *(genoset-class)*, 18
pos,RangedData-method *(genoset-class)*, 18
pos-methods *(pos)*, 25

RangedData, *18*
ranges *(genoset-methods)*, 19

ranges,-method *(genoset-methods)*, 19
ranges,GenoSet-method *(genoset-class)*, 18
rangeSampleMeans, 25
runCBS, 26

segment *(runCBS)*, 26
segmentation *(runCBS)*, 26
segs2Rle, 28
segs2RleDataFrame, 29
segTable, 28
show,GenoSet-method *(genoset-class)*, 18
space *(genoset-methods)*, 19
space,-method *(genoset-methods)*, 19
space,GenoSet-method *(genoset-class)*, 18
start *(genoset-methods)*, 19
start,-method *(genoset-methods)*, 19
start,GenoSet-method *(genoset-class)*, 18
subsetAssayData, 30

uniqueChrs, 31
uniqueChrs,-method *(uniqueChrs)*, 31
uniqueChrs,RangedDataOrGenoSet-method *(genoset-class)*, 18
uniqueChrs-methods *(uniqueChrs)*, 31
universe, 31
universe,-method *(universe)*, 31
universe,GenoSet-method *(genoset-class)*, 18
universe<- *(universe)*, 31
universe<-,-method *(universe)*, 31
universe<-,GenoSet-method *(genoset-class)*, 18