Package ‘genoset’

April 25, 2017

Type Package
Title A RangedSummarizedExperiment with methods for copy number analysis
Version 1.32.0
Date 2017-01-03
Author Peter M. Haverty
Maintainer Peter M. Haverty <phaverty@gene.com>
Description GenoSet provides an extension of the RangedSummarizedExperiment class with additional API features. This class provides convenient and fast methods for working with segmented genomic data. Additionally, GenoSet provides the class RleDataFrame which stores runs of data along the genome for multiple samples and provides very fast summaries of arbitrary row sets (regions of the genome).
License Artistic-2.0
LazyLoad yes
Depends R (>= 2.10), BiocGenerics (>= 0.11.3), GenomicRanges (>= 1.17.19), SummarizedExperiment (>= 1.1.6)
Imports S4Vectors (>= 0.13.13), GenomeInfoDb (>= 1.1.3), IRanges (>= 2.5.12), methods, graphics
Suggests RUnit, knitr, BiocStyle, rmarkdown, DNAcopy, stats, BSgenome, Biostrings
Enhances parallel
ByteCompile TRUE
biocViews Infrastructure, DataRepresentation, Microarray, SNP, CopyNumberVariation
Collate 'genoset-class.R' 'RleDataFrame-class.R'
'RleDataFrame-methods.R' 'bounds.R' 'ordering.R' 'plots.R'
'rangeSummaries.R' 'segments.R' 'test_genoset_package.R'
'utils.R'
VignetteBuilder knitr
URL https://github.com/phaverty/genoset
RoxygenNote 5.0.1
NeedsCompilation yes
### R topics documented:

<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>genoset-package</td>
<td>3</td>
</tr>
<tr>
<td>bat2mbaf</td>
<td>3</td>
</tr>
<tr>
<td>boundingIndices</td>
<td>4</td>
</tr>
<tr>
<td>boundingIndicesByChr</td>
<td>5</td>
</tr>
<tr>
<td>bounds2Rle</td>
<td>6</td>
</tr>
<tr>
<td>calcGC</td>
<td>6</td>
</tr>
<tr>
<td>calcGC2</td>
<td>7</td>
</tr>
<tr>
<td>chr</td>
<td>7</td>
</tr>
<tr>
<td>chrIndices</td>
<td>8</td>
</tr>
<tr>
<td>chrInfo</td>
<td>9</td>
</tr>
<tr>
<td>chrNames</td>
<td>9</td>
</tr>
<tr>
<td>chrOrder</td>
<td>10</td>
</tr>
<tr>
<td>chrPartitioning</td>
<td>11</td>
</tr>
<tr>
<td>cn2lr</td>
<td>11</td>
</tr>
<tr>
<td>fixSegNAs</td>
<td>12</td>
</tr>
<tr>
<td>gcCorrect</td>
<td>12</td>
</tr>
<tr>
<td>genome</td>
<td>13</td>
</tr>
<tr>
<td>genomeAxis</td>
<td>13</td>
</tr>
<tr>
<td>genoPlot</td>
<td>14</td>
</tr>
<tr>
<td>genoPos</td>
<td>15</td>
</tr>
<tr>
<td>GenoSet</td>
<td>16</td>
</tr>
<tr>
<td>GenoSet-class</td>
<td>17</td>
</tr>
<tr>
<td>genoset-datasets</td>
<td>18</td>
</tr>
<tr>
<td>isGenomeOrder</td>
<td>18</td>
</tr>
<tr>
<td>lr2cn</td>
<td>19</td>
</tr>
<tr>
<td>modeCenter</td>
<td>19</td>
</tr>
<tr>
<td>nrow,GenomicRanges-method</td>
<td>20</td>
</tr>
<tr>
<td>numCallable</td>
<td>20</td>
</tr>
<tr>
<td>pos,GenoSetOrGenomicRanges-method</td>
<td>21</td>
</tr>
<tr>
<td>rangeSampleMeans</td>
<td>21</td>
</tr>
<tr>
<td>rangeSegMeanLength</td>
<td>22</td>
</tr>
<tr>
<td>rbindDataframe</td>
<td>23</td>
</tr>
<tr>
<td>readGenoSet</td>
<td>23</td>
</tr>
<tr>
<td>RleDataFrame-class</td>
<td>24</td>
</tr>
<tr>
<td>RleDataFrame-views</td>
<td>26</td>
</tr>
<tr>
<td>runCBS</td>
<td>27</td>
</tr>
<tr>
<td>segPairTable</td>
<td>28</td>
</tr>
<tr>
<td>segs2Granges</td>
<td>30</td>
</tr>
<tr>
<td>segs2Rle</td>
<td>30</td>
</tr>
<tr>
<td>segs2RleDataFrame</td>
<td>31</td>
</tr>
<tr>
<td>segTable</td>
<td>32</td>
</tr>
<tr>
<td>toGenomeOrder</td>
<td>33</td>
</tr>
<tr>
<td>[,GenoSet,ANY-method</td>
<td>34</td>
</tr>
</tbody>
</table>

**Index**

35
**GenoSet: An eSet for data with genome locations**

**Description**

Load, manipulate, and plot copynumber and BAF data.

**See Also**

genoset-datasets

---

**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**

```r
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

**Examples**

```r
data(genoset,package="genoset")
baf = 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=list(rownames(genoset.ds),c("K","L")))
mbaf = baf2mbaf( genoset.ds[, , "baf"], hom.cutoff=0.9, calls = calls, call.pairs = list(K="L",L="L") ) # Sample L is matched normal for tumor sample K
m = baf2mbaf( genoset.ds[, , "baf"], hom.cutoff=0.9 ) # Put mbaf back into the BAFSet object
```

---

**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**

```r
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

**Examples**

```r
data(genoset,package="genoset")
baf = 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=list(rownames(genoset.ds),c("K","L")))
mbaf = baf2mbaf( genoset.ds[, , "baf"], hom.cutoff=0.9, calls = calls, call.pairs = list(K="L",L="L") ) # Sample L is matched normal for tumor sample K
m = baf2mbaf( genoset.ds[, , "baf"], hom.cutoff=0.9 ) # Put mbaf back into the BAFSet object
```
### 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 \(\leq\) start \(<\) stop \(\leq\) 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**

\[
\text{boundingIndices} (\text{starts, stops, positions, all.indices = FALSE})
\]

**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
- **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 \(O(k \times \log(n))\) generally and \(\sim O(k)\) 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. boundingIndices does not check for NAs or unsorted data in the subject positions. These assumptions are safe for position info coming from a GenoSet or GRanges.

**Value**

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

**See Also**

Other "range summaries": boundingIndicesByChr, rangeSampleMeans

**Examples**

\[
\text{starts} = \text{seq}(10, 100, 10)
\]

\[
\text{boundingIndices} (\text{starts=starts, stops=starts+5, positions = 1:100 })
\]
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 GenomicRanges

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

See Also

Other "range summaries": boundingIndices, 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

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

calcGC
Calculate GC Percentage in windows

Description
Local GC content can be used to remove GC artifacts from copynumber data (see Diskin et al, Nucleic Acids Research, 2008, PMID: 18784189). 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, bases = c("G", "C"))

Arguments
- object: GenomicRanges or GenoSet
- bsgenome: BSgenome, like Hsapiens from BSgenome.Hsapiens.UCSC.hg19 or DNAStringSet.
- expand: scalar integer, amount to expand each range before calculating gc
- bases: character, alphabet to count, usually c("G", "C"), but "N" is useful too
Value

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

Examples

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

calcGC2

Calculate GC Percentage in sliding window

Description

Local GC content can be used to remove GC artifacts from copynumber data (see Diskin et al, Nucleic Acids Research, 2008, PMID: 18784189). 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. Values are as.integer( 1e4 * fraction ) for space reasons.

Usage

calcGC2(dna)

Arguments

dna BSgenome or DNAStringSet

Value

SimpleRleList, integer 1e4 * GC fraction, chromosomes 1:22, X and Y

Examples

## Not run: library(BSgenome.Hsapiens.UCSC.hg19)
## Not run: gc = calcGC2(Hsapiens)

chr

Chromosome name for each feature

Description

Get chromosome name for each feature. Returns character.

Usage

chr(object)

## S4 method for signature 'GenoSet'
chr(object)

## S4 method for signature 'GenomicRanges'
chr(object)
chrIndices

Arguments

object GenoSet or GRanges

Value

character vector of chromosome positions for each feature

Examples

data(genoset, package = "genoset")
chr(genoset.ds) # c("chr1", "chr1", "chr1", "chr3", "chr3", "chrX", "chrX", "chrX", "chrX")
chr(rowRanges(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.

Usage

chrIndices(object, chr = NULL)

## S4 method for signature 'GenoSetOrGenomicRanges'
chrIndices(object, chr = NULL)

Arguments

object GenoSet or GRanges
chr character, specific chromosome name

Value

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

Examples

data(genoset, package = "genoset")
chrIndices(genoset.ds)
chrIndices(rowRanges(genoset.ds)) # The same
**chrInfo**

Get chromosome start and stop positions

**Description**

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

**Usage**

```r
chrInfo(object)
```

## S4 method for signature 'GenoSetOrGenomicRanges'

```r
chrInfo(object)
```

**Arguments**

- `object` A GenoSet object or similar

**Value**

list with start and stop position, by ordered chr

**Examples**

```r
data(genoset, package="genoset")
chrInfo(genoset.ds)
chrInfo(rowRanges(genoset.ds)) # The same
```

---

**chrNames**

Get list of unique chromosome names

**Description**

Get list of unique chromosome names

**Usage**

```r
chrNames(object)
```

## S4 method for signature 'GenoSet'

```r
chrNames(object)
```

## S4 method for signature 'GenomicRanges'

```r
chrNames(object)
```

```r
chrNames(object) <- value
```

## S4 replacement method for signature 'GenoSet'

```r
chrNames(object) <- value
```

## S4 replacement method for signature 'GenomicRanges'

```r
chrNames(object) <- value
```
### chrOrder

**Arguments**

- **object**: GenomicRanges or GenoSet
- **value**: return value of chrNames

**Value**

character vector with names of chromosomes

**Examples**

```r
data(genoset, package="genoset")
chrNames(genoset.ds) # c("chr1","chr3","chrX")
chrNames(rowRanges(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 ordered by number, then by letter.

**Usage**

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

**Arguments**

- **chr.names**: character, vector of unique chromosome names

**Value**

character vector of chromosome names in proper order

**See Also**

Other "genome ordering": isGenomeOrder, toGenomeOrder

**Examples**

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

**Partitioning by Chromosome**

**Description**
Get indices of first and last element in each chromosome.

**Usage**
chrPartitioning(object)

**Arguments**
- object: GenoSet or GenomicRanges

**Value**
PartitioningByEnd

---

**cn2lr**

**Take vector or matrix of copynumber values, convert to log2ratios**

**Description**
Utility function for converting copynumber units (2 is normal) to log2ratio units (two is normal). If ploidy is provided lr is log2(cn/ploidy), otherwise log2(cn/2).

**Usage**

```r
cn2lr(x, ploidy)
```

## S4 method for signature 'numeric'

cn2lr(x, ploidy)

## S4 method for signature 'matrix'

cn2lr(x, ploidy)

## S4 method for signature 'DataFrame'

cn2lr(x, ploidy)

**Arguments**
- x: numeric vector or matrix, or DataFrame with numeric-like columns (Rle typically). Assumed to be in copynumber units.
- ploidy: numeric, of length ncol(x). Ploidy of each sample.

**Value**
Data of same type as "x" transformed into log2ratio units
fixSegNA

**Fix NA runs in a Rle**

**Description**

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

**Usage**

```r
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

---

**See Also**

lr2cn

gcCorrect

**Correct copy number for GC content**

**Description**

Copy number estimates from various platforms show "Genomic Waves" (Diskin et al., Nucleic Acids Research, 2008, PMID: 18784189) 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**

```r
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

**Examples**

```r
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

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

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

Arguments

locs  GenomicRanges to be used to draw chromosome boundaries, if necessary. Usually rowRanges 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
genoPlot

Value
nothing

See Also
Other "genome plots": genoPlot

Examples

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

datat

Description
Plot location data and chromosome boundaries from a GenoSet 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.

Usage
genoPlot(x, y, ...)

## S4 method for signature 'numeric,numeric'
genoPlot(x, y, add = FALSE, xlab = "", ylab = "", col = "black", locs = NULL, ...)

## S4 method for signature 'numeric,Rle'
genoPlot(x, y, add = FALSE, xlab = "", ylab = "", col = "red", locs = NULL, lwd = 2, xlim = NULL, ...)

## S4 method for signature 'GenoSetOrGenomicRanges,ANY'
genoPlot(x, y, chr = NULL, add = FALSE, pch = ".", xlab = "", ylab = "", ...)

Arguments

  x    GenoSet (or descendant) or GRanges
  y    numeric or Rle
  ...  Additional plotting args
  add  Add plot to existing plot
Description

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

Usage

genoPos(object)

## S4 method for signature 'GenoSetOrGenomicRanges'
genoPos(object)
Arguments

object A GenoSet object or a GenomicRanges object

Value

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

Examples

data(geneset, package="genoset")
head(genoPos(geneset.ds))
head(genoPos(rowRanges(geneset.ds))) # The same

GenoSet Create a GenoSet object

Description

This function is the preferred method for creating a new GenoSet object. Currently, a GenoSet is simply a RangedSummarizedExperiment with some API changes and extra methods. Therefore, a GenoSet must always have a rowRanges.

Usage

GenoSet(rowRanges, assays, colData, metadata = list())

## S4 method for signature 'GenoSet'
lengths(x)

Arguments

rowRanges GenomicRanges, not a GenomicRangesList
assays list, SimpleList or matrix-like object
colData a data.frame or DataFrame of sample metadata with rownames matching the colnames of the matrices in assays
metadata a list of any other data you want to attach to the GenoSet object
x A GenoSet

Details

locations. Rownames are required to match featureNames.

Value

A GenoSet object
Examples

```r
test.sample.names = LETTERS[11:13]
probe.names = letters[1:10]
assays = list(matrix(31:60,nrow=10,ncol=3,dimnames=list(probe.names,test.sample.names)))
rowRanges = GRanges(ranges=IRanges(start=1:10,width=1,names=probe.names),seqnames=c(rep("chr1",4),rep("chr3",2),rep("chrX",4))
colData = data.frame(matrix(LETTERS[1:15],nrow=3,ncol=5,dimnames=list(test.sample.names,letters[1:5])))
rse = SummarizedExperiment(rowRanges=rowRanges,assays=assays,colData=colData,metadata=metadata)
gs = GenoSet(rowRanges, assays, colData)
```

Description

GenoSet class extends RangedSummarizedExperiment by adding some additional methods to the API. Examples include subsetting rows with a GenomicRanges and combining this with access to assays like genoset[i,j,assay].

Extends

Class RangedSummarizedExperiment, directly.

Methods

- `signature(x = "GenoSet", i = "ANY", j = "ANY", drop = "ANY")`
- `signature(x = "GenoSet", i = "character", j = "ANY", drop = "ANY")`
- `signature(x = "GenoSet", i = "ANY", j = "ANY", value = "ANY")`
- `signature(object = "GenoSet")`
- `signature(x = "GenoSet", y = "ANY")`
- `signature(x = "GenoSet")`
- `signature(x = "GenoSet")`
- `signature(x = "GenoSet")`
- `signature(x = "GenoSet")`
- `signature(x = "GenoSet")`
- `signature(x = "GenoSet")`
- `signature(x = "GenoSet", i="ANY")`
- `signature(x = "GenoSet", i="ANY",value="ANY")`
- `signature(x = "GenoSet")`
- `signature(x = "GenoSet")`
- `signature(x = "GenoSet")`
- `signature(x = "GenoSet")`
isGenomeOrder

See Also
GenoSet

Examples
```r
test.sample.names = LETTERS[11:13]
probe.names = letters[1:10]
assays=list(matrix(31:60,nrow=10,ncol=3,dimnames=list(probe.names,test.sample.names)))
rowRanges=GRanges(ranges=IRanges(start=1:10,width=1,names=probe.names),seqnames=c(rep("chr1",4),rep("chr3",2),rep("chrX",4)))
colData=data.frame(matrix(LETTERS[1:15],nrow=3,ncol=5,dimnames=list(test.sample.names,letters[1:5])))
rse=SummarizedExperiment(rowRanges=rowRanges,assays=assays,colData=colData)
gs = GenoSet(rowRanges, assays, colData)
```

genoset-datasets  Example GenoSet object

Description
A GenoSet object the 'baf' (B-Allele Frequency) and 'lrr' (Log-R Ratio) assay matrices. The 'lrr' assay matrix contains DNA copy number on the scale of tumor/ploidy and the 'baf' assay matrix contains data in the range 0 to 1 where 0 indicates the AA genotype, 0.5 indicates the AB genotype and 1 indicates the BB genotype.

Source
Simulated data

isGenomeOrder

Check if a GRanges or GenoSet 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 chromosome and start position only, rather than chromosome, strand, start, and width.

Usage
```r
isGenomeOrder(ds, strict = TRUE)
```

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

Value
logical
See Also
Other "genome ordering": chrOrder, toGenomeOrder

Examples

data(genoset, package="genoset")
isGenomeOrder(rowRanges(genoset.ds))

1r2cn

*Take vector or matrix of log2 ratios, convert to copynumber*

Description
Utility function for converting log2ratio units (zero is normal) to copynumber units (two is normal)

Usage
1r2cn(x)

Arguments
x numeric data in log2ratio values

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

See Also
cn2lr

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
dsn numeric matrix

Value
numeric matrix
Examples

```r
code1 <- matrix(rnorm(150, mean=0), ncol=3)
```

## S4 method for signature 'GenomicRanges' nrow(x)

Arguments

- `x`: A GenomicRanges

### numCallable

**Count Rle positions >= min**

Description

For Rle coverage vector, count number of positions where value >= min, think callable bases.

Usage

```r
numCallable(rle, bounds, min)
```

Arguments

- `rle`: integer Rle, no NAs
- `bounds`: IRanges or matrix, positions in Rle to consider. If bounds is a matrix, the first two columns are used as start and end.
- `min`: scalar integer, count Rle positions >= this value.

Value

integer vector of length nrow(bounds)
pos, GenoSetOrGenomicRanges-method

Chromosome position of features

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

Usage
```r
## S4 method for signature 'GenoSetOrGenomicRanges'
pos(x)
```

Arguments
- `x` GRanges GenoSet

Value
numeric vector of feature positions within a chromosome

Examples
```r
data(genoset, package="genoset")
pos(genoset.ds)  # 1:10
```

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. Range bounding is performed by the boundingIndices function.

Usage
```r
rangeSampleMeans(query, subject, assay.element, na.rm = FALSE)
```

Arguments
- `query` GRanges 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
- `na.rm` scalar logical, ignore NAs?
rangeSegMeanLength

Value
numeric matrix of features in each range averaged by sample

See Also
Other "range summaries": boundingIndicesByChr, boundingIndices

Examples

data(genoset)
my.genes = GRanges( ranges=IRanges(start=c(35e6,128e6),end=c(37e6,129e6),names=c("HER2","MYC")), seqnames=c("chr17","chr8") )
rangeSampleMeans( my.genes, genoset.ds, "lrr" )

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.

Usage
rangeSegMeanLength(range.gr, segs)

## S4 method for signature 'GRanges,list'
rangeSegMeanLength(range.gr, segs)

## S4 method for signature 'GRanges,data.frame'
rangeSegMeanLength(range.gr, segs)

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, segTable, segs2Granges, segs2RleDataFrame, segs2Rle
**rbindDataframe**  
*A fast method for concatenating data.frames*

**Description**  
Performs the same action as `do.call(rbind, list_of_dataframes)`, but dramatically faster. Part of the speed comes from assuming that all of the data.frames have the same column names and types. If desired, an additional factor column can be added that specifies the original list element associated with each row. The argument ‘element.colname’ is used to name this column.

**Usage**  
rbindDataframe(dflist, element.colname)

**Arguments**  
dflist  
list of data.frames  

element.colname  
scalar character, name for additional factor column giving the name of the element of ‘dflist’ corresponding to each row. ‘dflist’ must be named to use this feature.

**Details**  
For a list of 1000 data.frames with 884 rows and 12 columns ‘rbindDataframe’ takes 0.553s and ‘do.call(rbind,x)’ takes 327.304s, a 600X speedup. This pure-R solution is made possible by the lovely shallow copy features Michael Lawrence has added to base R.

**Value**  
data.frame

---

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

**Description**  
Given a rds file or a rda file with one GenoSet, load it, and return. Objects that pre-date the switch to a RangedSummarizedExperiment internal representation (V 1.29.0) are automatically switched to the new format.

**Usage**  
readGenoSet(path)

**Arguments**  
path  
character, path to rds or rda file
Value

GenoSet or related object (only object in RData file)

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")

RleDataFrame-class

Class "RleDataFrame"

Description

The RleDataFrame class serves to hold a collection of Run Length Encoded vectors (Rle objects) of the same length. For example, it could be used to hold information along the genome for a number of samples, such as sequencing coverage, DNA copy number, or GC content. This class inherits from both DataFrame and SimpleRleList (one of the AtomicVector types). This means that all of the usual subsetting and applying functions will work. Also, the AtomicList functions, like mean and sum, that automatically apply over the list elements will work. The scalar mathematical AtomicList methods can make this class behave much like a matrix (see Examples).

New objects can be created with the RleDataFrame constructor: RleDataFrame(..., row.names=NULL), where ... can be a list of Rle objects, or one or more individual Rle objects.

Use in Biobase eSet objects

The genoset class defines an annotatedDataFrameFrom method for DataFrame, which makes it possible to include DataFrames as assayData elements. The column names for DataFrame cannot be NULL, which makes it impossible to use them as assays in SummarizedExperiment at this time.

Row and Column Summaries

These objects will sometimes be in place of a matrix, as in the eSet example above. It is convenient to have some of the summarization methods for matrices. Each of these methods takes an RleDataFrame and returns a single Rle. The time required is similar to that required for a matrix. For an RleDataFrame x,

rowSums: Sum across 'rows'.

rowMeans: Means across 'rows'.

colSums: Sum each Rle. This is just the sum method for SimpleRleList.

colMeans: Mean of each Rle. This is just the mean method for SimpleRleList.

Slots

rownames: Object of class "character OR NULL" Names to describe each row of the DataFrame. These may end up taking more space than your collection of Rle objects, so consider leaving this NULL.

nrows: Object of class "integer" Number of rows.

elementType: Object of class "character" Notes that elements of the internal list are Rle objects.

elementMetadata: Object of class "DataTable OR NULL" Metadata on the elements, see DataFrame.

metadata: Object of class "list" Metadata on the whole object, see DataFrame.

listData: Object of class "list" Base list containing the Rle objects.
RleDataFrame-class

Extends

Class "SimpleRleList", directly. Class "DataFrame", directly.

Methods

as.matrix signature(x = "RleDataFrame"): Convert to matrix.
coerce signature(x = "RleDataFrame"): Convert to other classes.
colMeans signature(x = "RleDataFrame"): Mean of each column.
colSums signature(x = "RleDataFrame"): Sum of each column.
rowMeans signature(x = "RleDataFrame"): Mean of each 'row'.
rowSums signature(x = "RleDataFrame"): Sum of each 'row'.
show signature(object = "RleDataFrame"): Short and pretty description of an object of this type.

Author(s)

Peter M. Haverty, design suggestion from Michael Lawrence.

See Also

DataFrame AtomicList Rle RleList rowMeans colMeans rowSums colSums view-summarization-methods

Examples

class("RleDataFrame")

## Constructors
df = new("RleDataFrame", listData=list(A=Rle(c(NA, 2:3, NA, 5), rep(2, 5)), B=Rle(c(6:7, NA, 8:10),c(3,2,1,2,1,1)), nrows=10L))
df2 = RleDataFrame(list(A=Rle(c(NA, 2:3, NA, 5), rep(2, 5)), B=Rle(c(6:7, NA, 8:10),c(3,2,1,2,1,1))))
df3 = RleDataFrame(A=Rle(c(NA, 2:3, NA, 5), rep(2, 5)), B=Rle(c(6:7, NA, 8:10),c(3,2,1,2,1,1))))

## AtomicList Methods
runValue(df)
runLength(df)
ranges(df)
mean(df)
sum(df)
df + 5
log2(df) - 1

## Row and Column Summaries
rowSums(df)
colSums(df)
rowMeans(df)
colMeans(df)

## Coercion
as(df, "matrix")
as(df, "list")
as(df, "RleList")
as(df, "DataFrame")
as(df, "data.frame")

RleDataFrame-views Calculate summary statistics on views of an RleDataFrame

Description

These methods mirror the viewMeans type functions from IRanges for SimpleRleList. They differ in that they work on an RleDataFrame and an IRanges directly and also have a simplify argument. This works out to be faster (compute-wise) and also convenient.

Still, an RleDataFrame inherits from SimpleRleList, so all of the views functions will work.

Usage

rangeSums(x, bounds, na.rm=FALSE, simplify=TRUE)
rangeMeans(x, bounds, na.rm=FALSE, simplify=TRUE, ...)
rangeMins(x, bounds, na.rm=FALSE, simplify=TRUE)
rangeMaxs(x, bounds, na.rm=FALSE, simplify=TRUE)
rangeWhichMins(x, bounds, na.rm=FALSE, simplify=TRUE)
rangeWhichMaxs(x, bounds, na.rm=FALSE, simplify=TRUE)

Arguments

x RleDataFrame

bounds Matrix with two columns or IRanges representing ranges of rows of x to process. If bounds is a matrix, an IRanges is constructed assuming the first two columns represent the start and end of the ranges. The names for the IRanges is taken from the rownames of the matrix. Such a matrix can constructed with boundingIndicesByChr and is the preferred input.

na.rm Scalar logical. Ignore NAs in calculations?

simplify Scalar logical. Simplify result? If TRUE, the return value will be a vector or matrix. For a single view, a vector will be returned. Otherwise a matrix with one row per view and one column per column of x will be returned. If FALSE, the return value will be a list of length ncol(x) of vectors of length nrow(bounds).

... Additional arguments for other methods.

Details

The "range" name prefixes here serve to differentiate these functions from the "view" functions. This may change. I will be asking the IRanges team to add "..." and "simplify" to the "view" methods so that I can just make additional methods for RleDataFrame.

Value

With simplify == TRUE, a vector for single view or a matrix otherwise. When simplify == FALSE, a list of vectors length ncol(x) where each element is of length nrow(bounds).
runCBS

See Also
RleDataFrame boundingIndicesByChr

Examples

df = RleDataFrame(list(a=Rle(1:5, rep(2, 5))), b=Rle(1:5, rep(2, 5)),
row.names=LETTERS[1:10])
mat = matrix(c(1,4,3,5),ncol=2,dimnames=list(c("Gene1","Gene2"),c("start","end")))
bounds = IRanges(start=c(1, 4), end=c(3, 5), names=c("Gene1","Gene2"))

rangeMeans(df,bounds,simplify=FALSE)
rangeMeans(df,bounds,simplify=TRUE)
rangeMeans(df,mat,simplify=TRUE)
rangeMeans(df,bounds)
rangeSums(df,bounds)
rangeMins(df,bounds)
rangeMaxs(df,bounds)
rangeWhichMins(df,bounds)
rangeWhichMaxs(df,bounds)

# RleDataFrame isa SimpleRleList, so all the IRanges view* methods work too:
v = RleViewsList( lapply( df, Views, start=bounds ) )
viewMeans(v)

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 rowRanges 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
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.

Usage

segPairTable(x, y, ...)

## S4 method for signature 'Rle,Rle'
segPairTable(x, y, locs = NULL, chr.ind = NULL,
Arguments

- **x**: Rle or list/DataFrame of Rle vectors
- **y**: Rle or list/DataFrame of Rle vectors
- **...**: in generic, extra arguments for methods
- **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

See Also

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

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)
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, runCBS, segPairTable, segTable, segs2RleDataFrame, segs2Rle

Description
Take output of CBS, make Rle representing all features in ‘locs’ ranges. CBS output contains run length and run values for genomic segments, 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 rowRanges slot of a GenoSet
segs2RleDataFrame

Value

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

See Also

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

Examples

data(genoset, package="genoset")
segs = runCBS( genoset.ds[, , "lrr"], rowRanges(genoset.ds), return.segs=TRUE )
segs2Rle( segs[[1]], rowRanges(genoset.ds) )  # Take a data.frame of segments, say from DNAcopy's segment func

segs2RleDataFrame  CBS segments to probe matrix

Description

Given segments, make an RleDataFrame 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  rowRanges from a GenoSet object

Details

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

Value

RleDataFrame with n rows same as locs and one column for each sample

See Also

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

Examples

data(genoset, package="genoset")
seg.list = runCBS( genoset.ds[, , "lrr"], rowRanges(genoset.ds), return.segs=TRUE )
segs2RleDataFrame( seg.list, rowRanges(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 RleDataFrame and the rowRanges 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 rowRanges 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.

Usage

segTable(object, ...)

## S4 method for signature 'Rle'
segTable(object, locs = NULL, chr.ind = NULL,
         start = NULL, end = NULL, factor.chr = TRUE)

## S4 method for signature 'DataFrame'
segTable(object, locs, factor.chr = TRUE,
         stack = FALSE)

Arguments

object    Rle or RleDataFrame
...	       in generic, for extra args in methods
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

See Also

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

Examples

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

toGenomeOrder  Set a GRanges or GenoSet 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.

Usage

toGenomeOrder(ds, strict = TRUE)

Arguments

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

Details

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

Value

re-ordered ds

See Also

Other "genome ordering": chrOrder, isGenomeOrder

Examples

data(genoset, package="genoset")
toGenomeOrder( genoset.ds, strict=TRUE )
toGenomeOrder( genoset.ds, strict=FALSE )
toGenomeOrder( rowRanges(genoset.ds) )
[.GenoSet,ANY-method

Subset a GenoSet

Description
Subset a GenoSet

Usage
## S4 method for signature 'GenoSet,ANY'
x[i, j, k, ...., withDimnames = TRUE,
drop = FALSE]

## S4 replacement method for signature 'GenoSet,ANY,ANY,ANY'
x[i, j, k] <- value

Arguments
x          GenoSet
i          character, GRanges, logical, integer
j          character, logical, integer
k          character or integer
...        additional subsetting args
withDimnames scalar logical, put dimnames on returned assay?
drop       logical drop levels of space factor?
value      incoming data for assay "k", rows "i" and cols "j"

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

*Topic classes
  GenoSet-class, 17
  RleDataFrame-class, 24

*Topic datasets
genoset-datasets, 18

*Topic methods
  RleDataFrame-class, 24
  RleDataFrame-views, 26
  [\[,GenoSet,ANY-method, 34
  [\-,GenoSet,ANY,ANY,ANY-method
    (\[,GenoSet,ANY-method), 34
  as.matrix,RleDataFrame-method
    (RleDataFrame-class), 24

AtomicList, 25

baf2mbaf, 3
boundingIndices, 4, 5, 22
boundingIndicesByChr, 4, 5, 22, 27
bounds2Rle, 6, 22, 28–32

calcGC, 6
calcGC2, 7
chr, 7
chr,GenomicRanges-method (chr), 7
chr,GenoSet-method (chr), 7
chrIndices, 8
chrIndices,GenoSetOrGenomicRanges-method
  (chrIndices), 8
chrInfo, 9
chrInfo,GenoSetOrGenomicRanges-method
  (chrInfo), 9
chrNames, 9
chrNames,GenomicRanges-method
  (chrNames), 9
chrNames,GenoSet-method (chrNames), 9
chrNames<- (chrNames), 9
chrNames<-,GenomicRanges-method
  (chrNames), 9
chrNames<-,GenoSet-method (chrNames), 9
chrOrder, 10, 19, 33
chrPartitioning, 11
cn2lr, 11
cn2lr,DataFrame-method (cn2lr), 11
cn2lr,matrix-method (cn2lr), 11
cn2lr,numerical-method (cn2lr), 11
coerce,RleDataFrame,matrix-method
  (RleDataFrame-class), 24
colMeans, 25
colMeans,DataFrame-method
  (RleDataFrame-class), 24
colMeans,RleDataFrame-method
  (RleDataFrame-class), 24
colSums, 25
colSums,RleDataFrame-method
  (RleDataFrame-class), 24
dataFrame, 25

fixSegNAs, 12
gcCorrect, 12
genoPlot, 14

genoPlot,GenoSetOrGenomicRanges,ANY-method
  (genoPlot), 14
genoPlot,numeric,numeric-method
  (genoPlot), 14
genoPlot,numeric,Rle-method (genoPlot),
  14
genPos, 15
genPos,GenoSetOrGenomicRanges-method
  (genoPos), 15
GenoSet, 16, 18
genoset (genoset-package), 3
GenoSet-class, 17
genoset-datasets, 18
genoset-package, 3
genoset.ds (genoset-datasets), 18
GenoSetOrGenomicRanges-class
  (GenoSet-class), 17

isGenomeOrder, 10, 18, 33

lengths,GenoSet-method (GenoSet), 16
lr2cn, 19

modeCenter, 19

35
nrow, GenomicRanges-method, 20
numCallable, 20
pos, GenoSetOrGenomicRanges-method, 21
rangeColMeans (RleDataFrame-views), 26
RangedSummarizedExperiment, 17
rangeMaxs (RleDataFrame-views), 26
rangeMaxs, RleDataFrame-method (RleDataFrame-views), 26
rangeMeans (RleDataFrame-views), 26
rangeMeans, matrix-method (RleDataFrame-views), 26
rangeMeans, numeric-method (RleDataFrame-views), 26
rangeMeans, RleDataFrame-method (RleDataFrame-views), 26
rangeMins (RleDataFrame-views), 26
rangeMins, RleDataFrame-method (RleDataFrame-views), 26
rangeSampleMeans, 4, 5, 21
rangeSegMeanLength, 6, 22, 28–32
rangeSegMeanLength, GRanges, data.frame-method (rangeSegMeanLength), 22
rangeSegMeanLength, GRanges, list-method (rangeSegMeanLength), 22
rangeSums (RleDataFrame-views), 26
rangeSums, RleDataFrame-method (RleDataFrame-views), 26
rangeWhichMaxs (RleDataFrame-views), 26
rangeWhichMaxs, RleDataFrame-method (RleDataFrame-views), 26
rangeWhichMins (RleDataFrame-views), 26
rangeWhichMins, RleDataFrame-method (RleDataFrame-views), 26
rbindDataFrame, 23
readGenoSet, 23
Rle, 25
RleDataFrame, 27
RleDataFrame (RleDataFrame-class), 24
RleDataFrame-class, 24
RleDataFrame-views, 26
RleList, 25
rowMeans, 25
rowMeans, RleDataFrame-method (RleDataFrame-class), 24
rowSums, 25
rowSums, RleDataFrame-method (RleDataFrame-class), 24
runCBS, 6, 22, 27, 29–32
segPairTable, 6, 22, 28, 28, 30–32
segPairTable, DataFrame, DataFrame-method (segPairTable), 28
segPairTable, Rle, Rle-method (segPairTable), 28
segs2Granges, 6, 22, 28, 29, 30, 31, 32
segs2Rle, 6, 22, 28–30, 30, 31, 32
segs2RleDataFrame, 6, 22, 28–31, 31, 32
segTable, 6, 22, 28–31, 32
segTable, DataFrame-method (segTable), 32
segTable, Rle-method (segTable), 32
show, RleDataFrame-method (RleDataFrame-class), 24
SimpleRleList, 25
toGenomeOrder, 10, 19, 33