Package 'biovizBase'

November 12, 2026

Version 1.61.0

Title Basic graphic utilities for visualization of genomic data.

Description The biovizBase package is designed to provide a set of utilities, color schemes and conventions for genomic data. It serves as the base for various high-level packages for biological data visualization. This saves development effort and encourages consistency.

Depends R (>= 3.5.0), methods

Imports grDevices, stats, scales, Hmisc, RColorBrewer, dichromat, BiocGenerics, S4Vectors (>= 0.23.19), IRanges (>= 1.99.28), Seqinfo, GenomeInfoDb (>= 1.45.5), GenomicRanges (>= 1.61.1), SummarizedExperiment (>= 1.39.1), Biostrings (>= 2.77.2), Rsamtools (>= 2.25.1), GenomicAlignments (>= 1.45.1), GenomicFeatures (>= 1.61.4), AnnotationDbi, VariantAnnotation (>= 1.55.1), ensembldb (>= 2.33.1), AnnotationFilter (>= 0.99.8), rlang

Suggests BSgenome.Hsapiens.UCSC.hg19, TxDb.Hsapiens.UCSC.hg19.knownGene, BSgenome, rtracklayer, EnsDb.Hsapiens.v75, RUnit

License Artistic-2.0

LazyLoad Yes

LazyData Yes

Collate utils.R color.R AllGenerics.R crunch-method.R mold-method.R addStepping-method.R getFragLength-method.R shrinkageFun-method.R maxGap-method.R spliceSummary-method.R ideogram.R pileup.R coverage.R labs.R original.R transform.R facets-method.R aes.R scale.R zzz.R biovizBase-package.R

biocViews Infrastructure, Visualization, Preprocessing

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

git_branch devel

git_last_commit c17f233

git_last_commit_date 2026-04-28

Repository Bioconductor 3.24

Date/Publication 2026-11-12

Author Tengfei Yin [aut],
Michael Lawrence [aut, ths, cre],
Dianne Cook [aut, ths],
Johannes Rainer [ctb]

Maintainer Michael Lawrence <lawremi@gmail.com>

Contents

biovizBase-package . . . 3
addStepping-method . . . 3
colorBlindSafePal . . . 4
containLetters . . . 6
CRC . . . 7
crc1.GeRL . . . 7
crunch . . . 8
darned_hg19_subset500 . . . 9
estimateCoverage . . . 10
flatGrl . . . 10
GCcontent . . . 11
genesymbol . . . 12
getBioColor . . . 12
getFormalNames . . . 14
getGaps . . . 15
getIdeoGR . . . 16
getIdeogram . . . 17
getScale . . . 17
getXScale . . . 18
getYLab-method . . . 19
hg19Ideogram . . . 20
hg19IdeogramCyto . . . 20
ideo . . . 21
ideoCyto . . . 21
isIdeogram . . . 22
isMatchedWithModel . . . 22
isSimpleIdeogram . . . 23
maxGap-method . . . 24
mold . . . 25
parseArgsForAes . . . 26
pileupAsGRanges . . . 27
pileupGRangesAsVariantTable . . . 28
plotColorLegend . . . 29
showColor . . . 30
shrinkageFun-method . . . 30
splitByFacets . . . 31
strip_formula_dots . . . 33
subsetArgsByFormals . . . 33
transformGRangesForEvenSpace . . . 34
transformToGenome . . . 35

Index **38**

`biovizBase-package` *biovizBase is a package which provides utilities and color scheme...*

Description

biovizBase is a package which provides utilities and color scheme for higher level graphic package which aim to visualize biological data especially genetic data.

Details

This package provides default color scheme for nucleotide, strand, amino acid, try to pass colorblind checking as possible as we can. And also provide giemsa stain result color scheme used to show cytoband. This package also provides utilites to manipulate and summarize raw data to get them ready to be visualized.

`addStepping-method` *Adding disjoint levels to a GenomicRanges object*

Description

Adding disjoint levels to a GenomicRanges object

Usage

```
## S4 method for signature 'GenomicRanges'
addStepping(obj, group.name, extend.size = 0,
                        fix = "center",
                        group.selfish = TRUE)
```

Arguments

`obj`	A GenomicRanges object
`group.name`	Column name in the elementMetadata which specify the grouping information of all the entries. If provided, this will make sure all intervals belong to the same group will try to be on the same level and nothing falls in between.
`extend.size`	Adding invisible buffered region to the GenomicRanges object, if it's 10, then adding 5 at both end. This make the close neighbors assigned to the different levels and make your eyes easy to identify.
`fix`	"start", "end", or "center"(default) passed to `resize`, denoting what to use as an anchor for each element of obj, and add extend.size to it's current width.
`group.selfish`	Passed to `addStepping`, control whether to show each group as unique level or not. If set to `FALSE`, if two groups are not overlapped with each other, they will probably be layout in the same level to save space.

Details

This is a tricky question, for example, pair-end RNA-seq data could be represented as two set of GenomicRanges object, one indicates the read, one indicates the junction. At the same time, we need to make sure pair-ended read are shown on the same level, and nothing falls in between. For better visualization of the data, we may hope to add invisible extended buffer to the reads, so closely neighbored reads will be on the different levels.

Value

A modified GenmicRanges object with `stepping` as one column.

Author(s)

Tengfei Yin

Examples

```
library(GenomicRanges)
set.seed(1)
N <- 500
## sample GRanges
gr <- GRanges(seqnames =
              sample(c("chr1", "chr2", "chr3", "chrX", "chrY"),
                     size = N, replace = TRUE),
              IRanges(
                      start = sample(1:300, size = N, replace = TRUE),
                      width = sample(70:75, size = N,replace = TRUE)),
              strand = sample(c("+", "-", "*"), size = N,
                replace = TRUE),
              value = rnorm(N, 10, 3), score = rnorm(N, 100, 30),
              group = sample(c("Normal", "Tumor"),
                size = N, replace = TRUE),
              pair = sample(letters, size = N,
                replace = TRUE))

## grouping and extending
head(addStepping(gr))
head(addStepping(gr, group.name = "pair"))
gr.close <- GRanges(c("chr1", "chr1"), IRanges(c(10, 20), width = 9))
addStepping(gr.close)
addStepping(gr.close, extend.size = 5)
```

`colorBlindSafePal` *Color blind safe palette generator*

Description

This function help users create a function based on specified color blind safe palette. And the returned function could be used for color generation.

Usage

```
genDichromatPalInfo()
genBrewerBlindPalInfo()
genBlindPalInfo()
colorBlindSafePal(palette)
blind.pal.info
brewer.pal.blind.info
dichromat.pal.blind.info
```

Arguments

palette	A index numeric value or character. Please see `blind.pal.info`, the palette could be "pal_id" or names the row in which users could specify the palette you want to use.

Details

We get those color-blind safe palette based on http://colorbrewer2.org/ and http://geography.uoregon.edu/datagraphics/color_scales.htm those color are implemented in two packages, RColorBrewer and dichromat. But RColorBrewer doesn't provide subset of color-blind safe palette info. And dichromat doesn't group color based on "quality", "sequential" and "divergent", so we pick those color manually and create a combined palette, blind.pal.info.

`colorBlindSafePal` will return a function, this function will take two arguments, `n` and `repeatable`. if n is smaller than 3(n >= 3 is required by RColorBrewer), we use 3 instead and return a color vector. If n is over the `maxcolors` column in blind.pal.info, we use `maxcolors` instead when `repeatable` set to `FALSE`, if `repeatable` set to `TRUE` we repeat the color of all the allowed colors(length equals `maxcolors`) in the same order. This has special case in certain graphics which is always displayed side by side and don't worry about the repeated colors being neighbors.

`genBrewerBlindPalInfo` return `brewer.pal.blind.info` data frame containing all color-blind safe palettes from `brewer.pal.info` defined in RColorBrewer, but it's not only just subset of it, it also changes some `maxcolors` info.

`genDichromatPalInfo` return `dichromat.pal.blind.info` data frame.

`genBlindPalInfo` return `blind.pal.info` data frame.

Value

A color generating function with arguments `n` and `repeatable`. `n` specifying how many different discrete colors you want to get from them palette, and if repeatable turned on and set to TRUE, you can specify n even larger than maximum color. The color will be repeated following the same order.

Author(s)

Tengfei Yin

Examples

```
## Not run:
library(scales)
## brewer subse of only color blind palette
brewer.pal.blind.info
genBrewerBlindPalInfo()
## dichromat info
```

```
dichromat.pal.blind.info
genDichromatPalInfo()
## all color blind palette, adding id/pkg.
blind.pal.info
## with no parameters, just return blind.pal.info
colorBlindSafePal()
mypal <- colorBlindSafePal(20)
## or pass character name
mypal <- colorBlindSafePal("Set2")
mypal12 <- colorBlindSafePal(22)
show_col(mypal(12, repeatable = FALSE)) # warning
show_col(mypal(11, repeatable = TRUE))  # no warning, and repeat
show_col(mypal12(12))

## End(Not run)
```

`containLetters` *Checking if string contains letters or not*

Description

Test if a string contain any letters

Usage

```
containLetters(obj, all=FALSE)
```

Arguments

`obj`	String
`all`	If set to FALSE, return TRUE when any letters appears; if all is set to TRUE, return TRUE only when the string is composed of just letters.

Details

Useful when processing/sorting seqnames

Value

Logical value

Author(s)

tengfei

Examples

```
containLetters("XYZ123")
containLetters("XYZ123", TRUE)
```

CRC *CRC*

Description

Genomic sequencing of colorectal adenocarcinomas identifies a recurrent VTI1A-TCF7L2 fusion sample data set

Usage

```
data(CRC)
```

Details

The data used in this study is from this a paper http://www.nature.com/ng/journal/v43/n10/full/ng.936.html.

Examples

```
data(CRC)
```

crc1.GeRL *crc1.GeRL*

Description

CRC-1 mutation and structural rearrangment

Usage

```
data(crc1.GeRL)
```

Details

GenomicRangesList contains somatic mutation, reaarangment etc for a tumor sample, please check the reference for detials.

References

Genomic sequencing of colorectal adenocarcinomas identifies a recurrent VTI1A-TCF7L2 fusion

Examples

```
data(crc1.GeRL)
crc1.GeRL
```

crunch *Fetching GRanges from various data source*

Description

Fetching Granges from various data source, currently supported objects are TxDb, EnsDb, GAlignments and BamFile.

Usage

```
## S4 method for signature 'TxDb'
crunch(obj, which, columns = c("tx_id", "tx_name","gene_id"),
       type = c("all", "reduce"), truncate.gaps = FALSE,
       truncate.fun = NULL, ratio = 0.0025)
## S4 method for signature 'EnsDb'
crunch(obj, which, columns = c("tx_id", "gene_name","gene_id"),
       type = c("all", "reduce"), truncate.gaps = FALSE,
       truncate.fun = NULL, ratio = 0.0025)
## S4 method for signature 'GAlignments'
crunch(obj, which, truncate.gaps = FALSE,
       truncate.fun = NULL, ratio = 0.0025)
## S4 method for signature 'BamFile'
crunch(obj, which, ..., type = c("gapped.pair", "raw", "all"),
       truncate.gaps = FALSE, truncate.fun = NULL, ratio = 0.0025)
```

Arguments

obj	supported objects are TxDb, GAlignments and BamFile.
which	GRanges object. For TxDb object, could aslo be a list. For EnsDb, it can also be a single filter object extending AnnotationFilter-class, an AnnotationFilterList combining several such objects or a filter expression in form of a formula (see AnnotationFilter for examples).
columns	columns to include in the output.
type	default 'all' is to show the full model, 'reduce' is to show a single model.
truncate.gaps	logical value, default FALSE. Whether to truncate gaps or not.
truncate.fun	shrinkage function.
ratio	numeric value, shrinking ratio.
...	arguments passed to function readGAlignments.

Value

GRanges object.

Author(s)

Tengfei Yin

Examples

```
library(biovizBase)
library(TxDb.Hsapiens.UCSC.hg19.knownGene)
data(genesymbol, package = "biovizBase")
txdb <- TxDb.Hsapiens.UCSC.hg19.knownGene
obj <- txdb
temp <- crunch(txdb, which = genesymbol["BRCA1"], type = "all")
temp <- crunch(txdb, which = genesymbol["BRCA1"], type = "reduce")

## Fetching data from a EnsDb object.
library(EnsDb.Hsapiens.v75)
edb <- EnsDb.Hsapiens.v75
gr <- genesymbol["BRCA1"]
seqlevels(gr) <- sub(seqlevels(gr), pattern="chr", replacement="")
temp <- crunch(edb, which = gr)

## Alternatively, use the GenenameFilter from the AnnotationFilter package:
temp <- crunch(edb, which = GenenameFilter("BRCA1"))

## Or a filter expression
temp <- crunch(edb, which = ~ genename == "BRCA1")
```

---

darned_hg19_subset500 *Subset of RNA editing sites in hg19...*

Description

Subset of RNA editing sites in hg19

Usage

```
data(darned_hg19_subset500)
```

Details

This data set provides a subset(500 sites only) of hg19 RNA editing sites, and originally from DARNED http://darned.ucc.ie/ for the hg19 assembly.

Examples

```
data(darned_hg19_subset500)
darned_hg19_subset500
```

estimateCoverage *Estimation of Coverage*

Description

Estimation of Coverage

Usage

```
## S4 method for signature 'BamFile'
estimateCoverage(x, maxBinSize = 2^14)
```

Arguments

x	A BamFile object.
maxBinSize	Max bin size.

Value

A GRanges object.

Author(s)

Michael Lawrence

flatGrl *Transform GRangesList to GRanges*

Description

Transform GRangesList to GRanges.

Usage

```
flatGrl(object, indName = "grl_name")
```

Arguments

object	a GRangesList object.
indName	column named by 'indName' that groups the records by their original element in 'object'.

Details

This method is different from default stack, it integrate elementMetadata of GRangesList to the final coerced GRanges.

Value

A GRanges object.

Author(s)

Tengfei Yin

Examples

```
library(GenomicRanges)
gr1 <- GRanges("chr1", IRanges(1:10, width = 5))
gr2 <- GRanges("chr2", IRanges(1:10, width = 5))
obj <- GRangesList(gr1, gr2)
values(obj) <- data.frame(a = 1:2, b = letters[1:2])
stack(obj)
flatGrl(obj)
```

GCcontent *GC content computation for BSgenome*

Description

Compute GC content in a certain region of a BSgenome object

Usage

```
GCcontent(obj, ..., view.width, as.prob = TRUE)
```

Arguments

obj	BSgenome object
...	Arguments passed to getSeq method for BSgenome package.
view.width	Passed to letterFrequencyInSlidingView, the constant (e.g. 35, 48, 1000) size of the "window" to slide along obj. The specified letters are tabulated in each window of length view.width. The rows of the result (see value) correspond to the various windows.
as.prob	If TRUE return percentage of GC content, otherwise return counts.

Details

GC content is an interesting variable may be related to various biological questions. So we need a way to compute GC content in a certain region of a reference genome. GCcontent function is a wrapper around getSeq function in BSgenome package and letterFrequency, letterFrequencyInSlidingView in Biostrings package.

if the view.width is specified, the GC content will be computed in the sliding view

Value

Numeric value indicate count or percentage

Author(s)

Tengfei Yin

Examples

```
library(BSgenome.Hsapiens.UCSC.hg19)
GCcontent(Hsapiens, GRanges("chr1", IRanges(1e6, 1e6 + 1000)))
```

---

genesymbol *Gene symbols with position...*

Description

Gene symbols with position

Usage

```
data(genesymbol)
```

Details

This data set provides genen symbols in human with position and starnd information, stored as GRanges object.

Examples

```
data(genesymbol)
head(genesymbol)
genesymbol["RBM17"]
```

---

getBioColor *Color scheme getter for biological data*

Description

This function tries to get default color scheme either from fixed palette or options for specified data set, usually just biological data.

Usage

```
getBioColor(type = c("DNA_BASES_N", "DNA_BASES", "DNA_ALPHABET",
                     "RNA_BASES_N", "RNA_BASES", "RNA_ALPHABET",
                     "IUPAC_CODE_MAP", "AMINO_ACID_CODE", "AA_ALPHABET",
                     "STRAND", "CYTOBAND"),
            source = c("option", "default"))
```

Arguments

type	Color set based on which you want to get.
source	"option" tries to get color scheme from Options. This allow user to edit the color globally. "default" gets fixed color scheme.

Details

Most data set specified in the type argument are defined in Biostrings package, such as "DNA_BASES", "DNA_ALPHABET", "RNA_BASES", "RNA_ALPHABET", "IUPAC_CODE_MAP", "AMINO_ACID_CODE", "AA_ALPHABET", please check the manual for more details.

"DNA_BASES_N" is just "DNA_BASES" with extra "N" used in certain cases, like the result from `applyPileup` in Rsamtools package. We start with the five most used nucleotides, `A,T,C,G,N`, In genetics, `GC-content` usually has special biological significance because GC pair is bound by three hydrogen bonds instead of two like AT pairs. So it has higher thermostability which could result in different significance, like higher annealing temperature in PCR. So we hope to choose warm color for `G,C` and cold color for `A,T`, and a color in between to represent `N`. They are chosen from diverging color set of `color brewer`. So we should be able to easily tell the GC enriched region. This set of color also passed vischeck for colorblind people.

In `GRanges` object, we have `strand` which contains three levels, +, -, *. We are using a qualitative color set from `Color Brewer`. This color set pass the colorblind test. It should be a safe color set to use to color strand.

For most default color scheme if they are under 18, we are trying to use package dichromat to set color for color blind people. But for data set that contains more than 18 objects, it's not possible to assign colorblind-safe color to them anymore. So we need to repeat some color. It should not matter too much, because even normal people cannot tell the difference anymore.

Here are the definition for the data sets.

DNA_BASES Contains `A,C,T,G`.

DNA_ALPHABET This alphabet contains all letters from the IUPAC Extended Genetic Alphabet (see "?IUPAC_CODE_MAP") + the gap ("-") and the hard masking ("+") letters.

DNA_BASES_N Contains `A,C,T,G,N`

RNA_BASES_N Contains `A,C,U,G,N`

RNA_BASES Contains `A,C,T,G`

RNA_ALPHABET This alphabet contains all letters from the IUPAC Extended Genetic Alphabet (see ?IUPAC_CODE_MAP) where "T" is replaced by "U" + the gap ("-") and the hard masking ("+") letters.

IUPAC_CODE_MAP The "IUPAC_CODE_MAP" named character vector contains the mapping from the IUPAC nucleotide ambiguity codes to their meaning.

AMINO_ACID_CODE Single-Letter Amino Acid Code (see "?AMINO_ACID_CODE").

AA_ALPHABET This alphabet contains all letters from the Single-Letter Amino Acid Code (see "?AMINO_ACID_CODE") + the stop ("*"), the gap ("-") and the hard masking ("+") letters

STRAND Contains `"+", "-", "*"`

CYTOBAND Contains giemsa stain results:`gneg, gpos25, gpos50, gpos75,gpos100, gvar, stalk, acen`. Color defined in package geneplotter.

Value

A character of vector contains color(rgb), and the names of the vector is originally from the name of different data set. e.g. for DNA_BASES, it's just `A,C,T,G`. This allow users to get color for a vector of specified names. Please see the examples below.

Author(s)

Tengfei Yin

Examples

```
opts <- getOption("biovizBase")
opts$DNABasesNColor[1] <- "red"
options(biovizBase = opts)
## get from option(default)
getBioColor("DNA_BASES_N")
## get default fixed color
getBioColor("DNA_BASES_N", source = "default")
seqs <- c("A", "C", "T", "G", "G", "G", "C")
## get colors for a sequence.
getBioColor("DNA_BASES_N")[seqs]
```

---

getFormalNames *Get formals from functions*

Description

Get formals from functions, used for dispatching arguments inside.

Usage

```
getFormalNames(..., remove.dots = TRUE)
```

Arguments

`...`	functions.
`remove.dots`	logical value, indicate remove dots in formals or not, default is `TRUE`.

Value

A character vector for formal arguments.

Author(s)

Tengfei Yin

Examples

```
getFormalNames(plot, sapply)
```

getGaps *get gaps for a stepping transformed GRanges object*

Description

Get gaps for a stepping transformed GRanges object, for visualization purpose. So a extra "stepping" column is required. Please see details below for motivation.

Usage

```
## S4 method for signature 'GRanges'
getGaps(obj, group.name = NULL, facets = NULL)
```

Arguments

obj	A GRanges object.
group.name	group name, such as transcript ID, this is the group method within each panel of facets and gaps will be computed for each group of intervals.
facets	formula used for creating graphics, all variables must be present in the data.

Details

Since faceting is a subset and group process in visualization stage, some statistical computation need to be taken place after that. This leaves some computation like computing gaps hard based on solely GRanges object. Extra information like facets formula and group method would help to generate gaps which make sure they are aligned on the same level and within the same panel for grouped intervals. facets variables will be added to gaps GRanges along with group.name.

Value

A GRanges object representing gaps and for each row, the "stepping" column help later visualization and make sure gaps and intervals they are generated from are showed on the expected place.

Author(s)

Tengfei Yin

Examples

```
set.seed(1)
N <- 100
library(GenomicRanges)
gr <- GRanges(seqnames =
              sample(c("chr1", "chr2", "chr3"),
                     size = N, replace = TRUE),
              IRanges(
                      start = sample(1:300, size = N, replace = TRUE),
                      width = sample(70:75, size = N,replace = TRUE)),
              strand = sample(c("+", "-", "*"), size = N,
                replace = TRUE),
              value = rnorm(N, 10, 3), score = rnorm(N, 100, 30),
              sample = sample(c("Normal", "Tumor"),
```

```
                size = N, replace = TRUE),
              pair = sample(letters, size = N,
                replace = TRUE))

grl <- splitByFacets(gr, sample ~ seqnames)
gr <- unlist(endoapply(grl, addStepping))
gr.gaps <- getGaps(gr, group.name = "stepping", facets = sample ~ seqnames)
```

---

getIdeoGR *Get ideogram information*

Description

This function tries to parse ideogram information from seqlengths of a GRanges object. This information is usually used to plot chromosome background for kaytogram or esitmate proper lengths of chromosomes from data space for showing overview.

Usage

```
getIdeoGR(gr)
```

Arguments

gr	A GRanges object with or without lengths information.

Value

A ideogram GRanges object, each row indicate one single chromosome, and start with 1 and end with real chromosome length or estimated lengths.

Author(s)

Tengfei Yin

Examples

```
library(GenomicRanges)
data("hg19IdeogramCyto", package = "biovizBase")
hg19IdeogramCyto
## without seqlengths, simply reduce
getIdeoGR(hg19IdeogramCyto)
## with seqlengths
gr <- GRanges("chr1",  IRanges(1,3))
seqlevels(gr) <- c("chr1", "chr2", "chr3")
nms <- c(100, 200, 300)
names(nms) <- c("chr1", "chr2", "chr3")
seqlengths(gr) <- nms
gr
getIdeoGR(gr)
```

`getIdeogram` *Get ideogram.*

Description

Get ideogram w/o cytoband for certain genome

Usage

```
getIdeogram(genome, subchr, cytobands=TRUE)
```

Arguments

`genome`	Single specie names, which must be one of the result from `ucscGenomes()$db`. If missing, will invoke a menu for users to choose from.
`subchr`	A character vector used to subset the result.
`cytobands`	If TRUE, return ideogram with gieStain and name column. If FALSE, simply return the genome information for each chromosome.

Details

This function require a network connection, it will parse the data on the fly, function is a wrapper of some functionality from rtracklayer package to get certain table like cytoBand, a full table schema could be found `http://genome.ucsc.edu/cgi-bin/hgTables` in UCSC genome browser.This is useful for visualization of the whole genome or single chromosome, you can see some examples in ggbio package.

Value

A GRanges object.

Author(s)

Tengfei Yin

Examples

```
## Not run: hg19IdeogramCyto <- getIdeogram("hg19", cytoband = TRUE)
```

`getScale` *Get scale information from a GRanges*

Description

Trying to get scale information from a `GRanges` object, used for circular view for geom "scale".

Usage

```
getScale(gr, unit = NULL, n = 100, type = c("M", "B", "sci"))
```

Arguments

gr	a GRanges object.
unit	A numeric value for scale unit. Default NULL use argument n to estimate the unit.
n	Integer value to indicate how many scale ticks to make.
type	unit types to shown.

Value

A GRanges object, with extra column: "type" indicate it's longer major ticks or shorter minor ticks. "scale.y" indicates y height for major and minor ticks. Default ratio is 3:1.

Author(s)

Tengfei Yin

---

getXScale *get x scale breaks and labels*

Description

get x scale breaks and labels for GRanges with different coordintes(currently only "truncate_gaps" and "genome" supported).

Usage

```
 ## S4 method for signature 'GRanges'
getXScale(obj, type = c("default", "all", "left", "right"))
```

Arguments

obj	a GRanges object. "coord" in metadata shows proper coordinates transformation for this object.
type	types of labels for transformed data.

Value

list of breaks and labels.

Author(s)

Tengfei Yin

Examples

```
library(GenomicRanges)
gr1 <- GRanges("chr1", IRanges(start = c(100, 300, 600),
                               end = c(200, 400, 800)))
shrink.fun1 <- shrinkageFun(gaps(gr1), max.gap = maxGap(gaps(gr1), 0.15))
shrink.fun2 <- shrinkageFun(gaps(gr1), max.gap = 0)
s1 <- shrink.fun1(gr1)
getXScale(s1)
#  coord:genome
set.seed(1)
gr1 <- GRanges("chr1", IRanges(start = as.integer(runif(20, 1, 100)),
width = 5))
gr2 <- GRanges("chr2", IRanges(start = as.integer(runif(20, 1, 100)),
width = 5))
gr <- c(gr1, gr2)
gr.t <- transformToGenome(gr, space.skip = 1)
getXScale(gr.t)
```

---

getYLab-method *parse x and y label information from a specific object*

Description

parse y label information, object specific.

Usage

```
## S4 method for signature 'TxDb'
getYLab(obj)
## S4 method for signature 'GRanges'
getXLab(obj)
## S4 method for signature 'GRangesList'
getXLab(obj)
## S4 method for signature 'GAlignments'
getXLab(obj)
```

Arguments

obj	A TxDb object.

Value

a string.

Author(s)

Tengfei Yin

hg19Ideogram *Hg19 ideogram without cytoband information...*

Description

Hg19 ideogram without cytoband information

Usage

```
data(hg19Ideogram)
```

Details

This data set provides hg19 genome information wihout cytoband information.

Examples

```
data(hg19Ideogram)
hg19Ideogram
```

hg19IdeogramCyto *Hg19 ideogram with cytoband information...*

Description

Hg19 ideogram with cytoband information

Usage

```
data(hg19IdeogramCyto)
```

Details

This data set provides hg19 genome information with cytoband information.

Examples

```
data(hg19IdeogramCyto)
hg19IdeogramCyto
```

ideo *ideogram without cytoband information*

Description

ideogram without cytoband information

Usage

```
data(ideo)
```

Details

This data set provides hg19, hg18, mm10, mm9 genome information wihout cytoband information as a lit.

Examples

```
data(ideo)
ideo
```

ideoCyto *ideogram with cytoband information*

Description

ideogram with cytoband information

Usage

```
data(ideoCyto)
```

Details

This data set provides hg19, hg18, mm10, mm9 genome information with cytoband information as a lit.

Examples

```
data(ideoCyto)
ideoCyto
```

isIdeogram *Ideogram checking*

Description

Check if an object is ideogram or not

Usage

```
isIdeogram(obj)
```

Arguments

obj	object

Details

Simply test if it's the result coming from `getIdeogram` function or not, make sure it's a `GRanges` and with extra column

Value

A logical value to indicate it's a ideogram or not.

Author(s)

Tengfei Yin

Examples

```
data(hg19IdeogramCyto)
data(hg19Ideogram)
isIdeogram(hg19IdeogramCyto)
isIdeogram(hg19Ideogram)
```

isMatchedWithModel *Utils for Splicing Summary*

Description

Utilities used for summarizing isoforms

Usage

```
isJunctionRead(cigar)
isMatchedWithModel(model, gr)
```

Arguments

cigar	A CIGAR string vector.
model	A GRanges object.
gr	A GRanges object.

Details

isJunctionRead simply parsing the CIGAR string to see if there is "N" in between and return a logical vector of the same length as cigar parameters, indicate it's junction read or not.

isMatchedWithModel mapping gr to model, and counting overlapped cases for each row of model, If gr contains all the read, this will return a logical vector of the same length as gr, and indicate if each read is the support for this model. NOTICE: we only assume it's a full model, so each model here is simply one isoform. So we only treat the gaped reads which only overlapped with two consecutive exons in model as one support for it.

Value

Logical vectors.

Author(s)

Tengfei Yin

Examples

```
library(GenomicAlignments)
bamfile <- system.file("extdata", "SRR027894subRBM17.bam",
  package="biovizBase")
## get index of junction read
which(isJunctionRead(cigar(readGAlignments(bamfile))))
##
model <- GRanges("chr1", IRanges(c(10, 20, 30, 40), width  = 5))
gr <- GRanges("chr1", IRanges(c(10, 10, 12, 22, 33), c(31, 40, 22, 32,
  44)))
isMatchedWithModel(model, gr)
```

---

isSimpleIdeogram *Simple ideogram checking*

Description

Check if an object is a simple ideogram or not

Usage

```
isSimpleIdeogram(obj)
```

Arguments

obj	object

Details

test if it's GRanges or not, doesn't require cytoband information. But it double check to see if there is only one entry per chromosome

Value

A logical value to indicate it's a simple ideogram or not.

Author(s)

tengfei

Examples

```
data(hg19IdeogramCyto)
data(hg19Ideogram)
isSimpleIdeogram(hg19IdeogramCyto)
isSimpleIdeogram(hg19Ideogram)
```

`maxGap-method` *Estimated max gaps*

Description

Compute an estimated max gap information, which could be used as max.gap argument in `shringkageFun`.

Usage

```
## S4 method for signature 'GenomicRanges'
maxGap(obj, ratio = 0.0025)
```

Arguments

`obj`	GenomicRanges object
`ratio`	Multiple by the range of the provided gaps as the max gap.

Details

This function tries to estimate an appropriate max gap to be used for creating a shrinkage function.

Value

A numeric value

Author(s)

Tengfei Yin

See Also

`shrinkageFun`

Examples

```
require(GenomicRanges)
gr1 <- GRanges("chr1", IRanges(start = c(100, 300, 600),
end = c(200, 400, 800)))
gr2 <- GRanges("chr1", IRanges(start = c(100, 350, 550),
end = c(220, 500, 900)))
gaps.gr <- intersect(gaps(gr1, start = min(start(gr1))),
gaps(gr2, start = min(start(gr2))))
shrink.fun <- shrinkageFun(gaps.gr, max.gap = maxGap(gaps.gr))
shrink.fun(gr1)
shrink.fun(gr2)
```

`mold` *mold data into data.frame*

Description

mold data into data.frame usued for visualization, so it's sometimes not as simple as coersion, for example, we add midpoint varialbe for mapping.

Usage

```
## S4 method for signature 'IRanges'
mold(data)
## S4 method for signature 'GRanges'
mold(data)
## S4 method for signature 'GRangesList'
mold(data,indName = "grl_name")
## S4 method for signature 'Seqinfo'
mold(data)
## S4 method for signature 'matrix'
mold(data)
## S4 method for signature 'eSet'
mold(data)
## S4 method for signature 'ExpressionSet'
mold(data)
## S4 method for signature 'RangedSummarizedExperiment'
mold(data, assay.id = 1)
## S4 method for signature 'Views'
mold(data)
## S4 method for signature 'Rle'
mold(data)
## S4 method for signature 'RleList'
mold(data)
## S4 method for signature 'VRanges'
mold(data)
```

Arguments

`data`	data to be mold into data.frame with additional column that helped mapping.For example we add 'midpoint' variable to a ranged converted data.frame.

indName	when collapsing a GRangesList, list names are aggregated into extra column named by this parameter.
assay.id	define the assay id you want to convert into a data.frame.

Value

a data.frame object.

Author(s)

Tengfei Yin

---

parseArgsForAes *Utils for parsing (un)evaluated arguments*

Description

Utilities for parsing (un)evaluated arguments

Usage

```
parseArgsForAes(args)
parseArgsForNonAes(args)
```

Arguments

args	arguments list.

Value

For parseArgsForAes return a unevaluated arguments.

For parseArgsNonForAes return a evaluated/quoted arguments.

Author(s)

Tengfei Yin

Examples

```
args <- alist(a = color, b = "b")
# parseArgsForAes(args)
```

pileupAsGRanges *Summarize reads for certain region*

Description

This function summarize reads from bam files for nucleotides on single base unit in a given region, this allows the downstream mismatch summary analysis.

Usage

```
pileupAsGRanges(bams, regions, DNABases=c("A", "C", "G", "T", "N"), ...)
```

Arguments

bams	A character which specify the bam file path.
regions	A GRanges object specifying the region to be summarized. This passed to which arguments in ApplyPileupsParam.
DNABases	Nucleotide type you want to summarize in the result and in specified order. It must be one or more of A,C,G,T,N.
...	Extra parameters passed to ApplyPileupsParam.

Details

It's a wrapper around applyPileup function in Rsamtools package, more detailed control could be found under manual of ApplyPileupsParam function in Rsamtools. pileupAsGRanges function return a GRanges object which including summary of nucleotides, depth, bam file path. This object could be read directly into pileupGRangesAsVariantTable function for mismatch summary.

Value

A GRanges object, each row is one single base unit. and elementMetadata contains summary about this position about all nucleotides specified by DNABases. and depth for total reads, bam for file path.

Author(s)

Michael Lawrence, Tengfei Yin

Examples

```
## Not run:
library(Rsamtools)
data(genesymbol)
library(BSgenome.Hsapiens.UCSC.hg19)
bamfile <- system.file("extdata", "SRR027894subRBM17.bam", package="biovizBase")
test <- pileupAsGRanges(bamfile, region = genesymbol["RBM17"])
test.match <- pileupGRangesAsVariantTable(test, Hsapiens)
head(test[,-7])
head(test.match[,-5])

## End(Not run)
```

pileupGRangesAsVariantTable
Mismatch summary

Description

Compare to reference genome and compute mismatch summary for certain region of reads.

Usage

```
pileupGRangesAsVariantTable(gr, genome, DNABases=c("A", "C", "G", "T", "N"))
```

Arguments

gr	A GRanges object, with nucleotides summary, each base take one column in elementMetadata or user can simply passed the returned result from pileupAsGRanges function to this function.
genome	BSgenome object, need to be the reference genome.
DNABases	Nucleotide types contained in passed GRanges object. Default is A/C/G/T/N, it tries to match the column names in elementMetadata to those default nucleotides. And treat the matched column as base names.

Details

User need to make sure to pass the right reference genome to this function to get the right summary. This function drop the position has no reads and only keep the region with coverage in the summary. The result could be used to show stacked barchart for mismatch summary.

Value

A GRanges object. Containing the following elementMetadata

- refNucleotide in reference genome.
- readNucleotide contained in the reads at particular position, if multiple nucleotide, either matched or unmatched are found, they will be summarized in different rows.
- countCount for read column.
- matchLogical value, whether matched to reference genome or not
- bamCharacter indicate bam file path.

Author(s)

Michael Lawrence, Tengfei Yin

Examples

```
## Not run:
library(Rsamtools)
data(genesymbol)
library(BSgenome.Hsapiens.UCSC.hg19)
bamfile <- system.file("extdata", "SRR027894subRBM17.bam", package="biovizBase")
test <- pileupAsGRanges(bamfile, region = genesymbol["RBM17"])
```

```
test.match <- pileupGRangesAsVariantTable(test, Hsapiens)
head(test[,-7])
head(test.match[,-5])

## End(Not run)
```

`plotColorLegend` *Show colors*

Description

Plot color legend, simple way to check your default color scheme

Usage

```
plotColorLegend(colors, labels, title)
```

Arguments

`colors`	A character vector of colors
`labels`	Labels to put aside colors, if `missing`, use names of the colors character vector.
`title`	Title for the color legend.

Details

Show color sheme as a legend style, labels

Value

A graphic device showing color legend.

Author(s)

Tengfei Yin

Examples

```
cols <- getOption("biovizBase")$baseColor
plotColorLegend(cols, title = "strand legend")
```

showColor *Show colors*

Description

Show colors with color string or names of color vectors.

Usage

```
showColor(colors, label = c("color", "name"))
```

Arguments

colors	A color character vector.
label	"color" label color with simple color string, and "name" label color with names of the vectors.

Value

A plot.

Author(s)

Tengfei Yin

Examples

```
## Not run:
showColor(getBioColor("CYTOBAND"))

## End(Not run)
```

shrinkageFun-method *Shrinkage function*

Description

Create a shrinkage function based on specified gaps and shrinkage rate.

Usage

```
## For IRanges
## S4 method for signature 'IRanges'
shrinkageFun(obj, max.gap = 1L)

## For GenomicRanges
## S4 method for signature 'GenomicRanges'
shrinkageFun(obj, max.gap = 1L)

is_coord_truncate_gaps(obj)
```

Arguments

obj	GenomicRanges object which represent gaps
max.gap	Gaps to be kept, it's a fixed value, if this value is bigger than certain gap interval, then that gap is not going to be shrunk.

Details

shrinkageFun function will read in a GenomicRanges object which represent the gaps, and return a function which works for another GenomicRanges object, to shrink that object based on previously specified gaps shrinking information. You could use this function to treat multiple tracks(e.g. GRanges) to make sure they shrunk based on the common gaps and the same ratio.

is_coord_truncate_gaps is used to check if a GRanges object is in "truncate_gaps" coordiantes or not.

Value

A shrinkage function which could shrink a GenomicRanges object, this function will add coord "truncate_gaps" and max.gap to metadata, ".ori" for oringal data as extra column

Author(s)

Michael Lawrence, Tengfei Yin

Examples

```
library(GenomicRanges)
gr1 <- GRanges("chr1", IRanges(start = c(100, 300, 600),
                               end = c(200, 400, 800)))

shrink.fun1 <- shrinkageFun(gaps(gr1), max.gap = maxGap(gaps(gr1), 0.1))
shrink.fun2 <- shrinkageFun(gaps(gr1, start(gr1), end(gr1)), max.gap = maxGap(gaps(gr1), 0.1))
shrink.fun3 <- shrinkageFun(gaps(gr1), max.gap = 0)
s1 <- shrink.fun1(gr1)
s2 <- shrink.fun2(gr1)
s3 <- shrink.fun3(gr1)
metadata(s1)$coord
values(s1)$.ori
is_coord_truncate_gaps(s1)
```

---

splitByFacets *split a GRanges by formula*

Description

Split a GRanges by formula into GRangesList. Parse variables in formula and form a interaction factors then split the GRanges by the factos.

Usage

```
## S4 method for signature 'GRanges,formula'
splitByFacets(object, facets)
## S4 method for signature 'GRanges,GRanges'
splitByFacets(object, facets)
## S4 method for signature 'GRanges,NULL'
splitByFacets(object, facets)
## S4 method for signature 'GRanges,missing'
splitByFacets(object, facets)
```

Arguments

object	GRanges object.
facets	formula object, such as . ~ seqnames. Or GRanges object, or NULL.

Details

if facets is formula, factors are crreated based on interaction of variables in formula, then split it with this factor. If facets is GRanges object, it first subset the original data by facets GRanges, then split by each region in the facets. If facets is NULL, split just by seqnames. This function is used to perform computation in conjunction with facets argments in higher level graphic function.

Value

A GRangesList.

Author(s)

Tengfei Yin

Examples

```
library(GenomicRanges)
N <- 1000
gr <- GRanges(seqnames =
              sample(c("chr1", "chr2", "chr3"),
                     size = N, replace = TRUE),
              IRanges(
                      start = sample(1:300, size = N, replace = TRUE),
                      width = sample(70:75, size = N,replace = TRUE)),
              strand = sample(c("+", "-", "*"), size = N,
                replace = TRUE),
              value = rnorm(N, 10, 3), score = rnorm(N, 100, 30),
              sample = sample(c("Normal", "Tumor"),
                size = N, replace = TRUE),
              pair = sample(letters, size = N,
                replace = TRUE))
facets <- sample ~ seqnames
splitByFacets(gr, facets)
splitByFacets(gr)
gr.sub <- GRanges("chr1", IRanges(c(1, 200, 250), width = c(50, 10,
30)))
splitByFacets(gr, gr.sub)
```

strip_formula_dots *strip dots around a formula variables*

Description

strip dots around variables in a formula.

Usage

```
strip_formula_dots(formula)
```

Arguments

formula	A formula. such as coverage ~ seqnames, or ..coverage.. ~ seqnames.

Value

A formula wihout ".." around for all variables.

Author(s)

Tengfei Yin

Examples

```
obj <- ..coverage.. ~ seqnames
strip_formula_dots(obj)
```

subsetArgsByFormals *Subset list of arguments by functions*

Description

find arguments matched by formals of passed functions,

Usage

```
subsetArgsByFormals(args, ..., remove.dots = TRUE)
```

Arguments

args	list of arguments with names indicate the formals.
...	functions used to parse formals.
remove.dots	logical value indicate whether to include dots in formals or not.

Value

argumnets that matched with passed function.

Author(s)

Tengfei Yin

Examples

```
args <- list(x = 1:3, simplify = TRUE, b = "b")
subsetArgsByFormals(args, plot, sapply)
```

---

transformGRangesForEvenSpace

Transform GRanges with New Coordinates

Description

For graphics, like linked plot, e.g. generated by `qplotRangesLinkedToData` function in package ggbio. we need to generate a new set of coordinates which is used for even spaced statistics track.

Usage

```
transformGRangesForEvenSpace(gr)
```

Arguments

gr	A GRanges object.

Details

Most used internally for special graphics, like `qplotRangesLinkedToData` function in package ggbio.

Value

A GRanges object as passed in, with new column x.new which indicate the static track coordinates, in this way, we could map the new coordinates with the old one.

Author(s)

Tengfei Yin

Examples

```
library(GenomicRanges)
gr <- GRanges("chr1", IRanges(seq(1,100, length.out = 10), width = 5))
library(biovizBase)
transformGRangesForEvenSpace(gr)
```

transformToGenome *Transform GRanges to different coordinates and layout*

Description

Used for coordiante genome transformation, other transformation in circular view.

Usage

```
## S4 method for signature 'GRanges'
transformToGenome(data, space.skip = 0.1, chr.weight
= NULL)
## S4 method for signature 'GRangesList'
transformToGenome(data, space.skip = 0.1,
chr.weight = NULL)

## S4 method for signature 'GRanges'
transformToArch(data, width = 1)
transformToCircle(data, x = NULL, y = NULL, ylim = NULL,
                  radius = 10, trackWidth =10,
                  direction = c("clockwise", "anticlockwise"),
                   mul = 0.05)


transformToRectInCircle(data, y = NULL, space.skip = 0.1, trackWidth = 10, radius = 10,
                      direction = c("clockwise", "anticlockwise"),
                      n = 100, mul = 0.05, chr.weight = NULL)

transformToBarInCircle(data, y = NULL, space.skip = 0.1, trackWidth = 10, radius = 10,
                     direction = c("clockwise", "anticlockwise"),
                      n = 100, mul = 0.05, chr.weight = NULL)

transformToSegInCircle(data, y = NULL, space.skip = 0.1, trackWidth = 10, radius = 10,
                      direction = c("clockwise", "anticlockwise"), n =
                      100, chr.weight = NULL)


transformToLinkInCircle(data, linked.to, space.skip = 0.1, trackWidth = 10, radius = 10,
                  link.fun = function(x, y, n = 100) bezier(x, y, evaluation = n),
                  direction = c("clockwise", "anticlockwise"), chr.weight = NULL)

transformDfToGr(data, seqnames = NULL, start = NULL, end = NULL,
                            width = NULL, strand = NULL,
                            to.seqnames = NULL, to.start = NULL, to.end = NULL,
                            to.width = NULL, to.strand = NULL, linked.to
                            = to.gr)



## S4 method for signature 'GRanges'
transformToDf(data)
```

```
is_coord_genome(data)
```

Arguments

data	a GRanges object. for function transformDfToGr it's data.frame.
x	character for variable as x axis used for transformation.
y	character for variable as y axis used for transformation.
ylim	numeric range to control the ylimits on tracks when 'y' information is involved.
space.skip	numeric values indicates skipped ratio of whole space, not skipped space is identical between each space.
radius	numeric value, indicates radius when transform to a circle.
trackWidth	numeric value, for track width.
direction	"clockwise" or "counterclockwise", for layout or transform direction to circle.
mul	numeric value, passed to expand_range function, to control margin of y in the track.
n	integer value, control interpolated points numbers.
linked.to	a column name of GRanges object, indicate the linked line's end point which represented as a GRanges too..
link.fun	function used to generate linking lines.
seqnames	character or integer values for column name or index indicate variable mapped to seqnames, default NULL use "seqnames".
start	character or integer values for column name or index indicate variable mapped to start, default NULL use "start".
end	character or integer values for column name or index indicate variable mapped to end, default NULL use "end".
width	character or integer values for column name or index indicate variable mapped to width, default NULL use "width".
strand	character or integer values for column name or index indicate variable mapped to strand, default NULL use "strand".
to.seqnames	character or integer values for column name or index indicate variable mapped to linked seqnames, default NULL, create GRanges without new GRanges attached as column. If this varialbe is not NULL, this mean you try to parse linked GRanges object.
to.start	character or integer values for column name or index indicate variable mapped to start of linked GRanges, default NULL use "to.start".
to.end	character or integer values for column name or index indicate variable mapped to end of linked GRanges, default NULL use "to.end".
to.width	character or integer values for column name or index indicate variable mapped to width of linked GRanges, default NULL use "to.width".
to.strand	character or integer values for column name or index indicate variable mapped to strand, default NULL use "to.strand" or just use *.
chr.weight	numeric vectors which sum to <1, the names of vectors has to be matched with seqnames in seqinfo, and you can only specify part of the seqnames, other lengths of chromosomes will be assined proportionally to their seqlengths, for example, you could specify chr1 to be 0.5, so the chr1 will take half of the space and other chromosomes squeezed to take left of the space.

Value

A GRanges object, with calculated new variables, including ".circle.x" for transformed x position, ".circle.y" for transformed y position, ".circle.angle" for transformed angle.

Author(s)

Tengfei Yin

Examples

```
library(biovizBase)
library(GenomicRanges)
set.seed(1)
gr1 <- GRanges("chr1", IRanges(start = as.integer(runif(20, 1, 2e9)),
width = 5))
gr2 <- GRanges("chr2", IRanges(start = as.integer(runif(20, 1, 2e9)),
width = 5))
gr <- c(gr1, gr2)
gr.t <- transformToGenome(gr, space.skip = 0.1)
is_coord_genome(gr.t)
transformToCircle(gr.t)
transformToRectInCircle(gr)
transformToSegInCircle(gr)
values(gr1)$to.gr <- gr2
transformToLinkInCircle(gr1, linked.to = "to.gr")
```

Index

∗ **datasets**
CRC, 7
crc1.GeRL, 7
darned_hg19_subset500, 9
genesymbol, 12
hg19Ideogram, 20
hg19IdeogramCyto, 20
ideo, 21
ideoCyto, 21

addStepping *(addStepping-method)*, 3
addStepping,GenomicRanges-method *(addStepping-method)*, 3
addStepping-method, 3
AnnotationFilter, *8*
AnnotationFilterList, *8*

biovizBase *(biovizBase-package)*, 3
biovizBase-package, 3
blind.pal.info *(colorBlindSafePal)*, 4
brewer.pal.blind.info *(colorBlindSafePal)*, 4

colorBlindSafePal, 4
containLetters, 6
CRC, 7
crc.gr *(CRC)*, 7
crc1.GeRL, 7
crunch, 8
crunch,BamFile-method *(crunch)*, 8
crunch,EnsDb-method *(crunch)*, 8
crunch,GAlignments-method *(crunch)*, 8
crunch,TxDb-method *(crunch)*, 8

darned_hg19_subset500, 9
data.frame, *25, 26*
dichromat.pal.blind.info *(colorBlindSafePal)*, 4

estimateCoverage, 10
estimateCoverage,BamFile-method *(estimateCoverage)*, 10

flatGrl, 10

GCcontent, 11
genBlindPalInfo *(colorBlindSafePal)*, 4
genBrewerBlindPalInfo *(colorBlindSafePal)*, 4
genDichromatPalInfo *(colorBlindSafePal)*, 4
genesymbol, 12
getBioColor, 12
getFormalNames, 14
getGaps, 15
getGaps,GRanges-method *(getGaps)*, 15
getIdeoGR, 16
getIdeogram, 17
getScale, 17
getXLab *(getYLab-method)*, 19
getXLab,GAlignments-method *(getYLab-method)*, 19
getXLab,GRanges-method *(getYLab-method)*, 19
getXLab,GRangesList-method *(getYLab-method)*, 19
getXLab-method *(getYLab-method)*, 19
getXScale, 18
getXScale,GRanges-method *(getXScale)*, 18
getYLab *(getYLab-method)*, 19
getYLab,TxDb-method *(getYLab-method)*, 19
getYLab-method, 19
GRangesList, *26*

hg19Ideogram, 20
hg19IdeogramCyto, 20
hg19sub *(CRC)*, 7

ideo, 21
ideoCyto, 21
is_coord_genome *(transformToGenome)*, 35
is_coord_truncate_gaps *(shrinkageFun-method)*, 30
isIdeogram, 22
isJunctionRead *(isMatchedWithModel)*, 22
isMatchedWithModel, 22
isSimpleIdeogram, 23

maxGap *(maxGap-method)*, 24

maxGap,GenomicRanges-method
(maxGap-method), 24
maxGap-method, 24
mold, 25
mold,eSet-method (mold), 25
mold,ExpressionSet-method (mold), 25
mold,GRanges-method (mold), 25
mold,GRangesList-method (mold), 25
mold,IRanges-method (mold), 25
mold,matrix-method (mold), 25
mold,RangedSummarizedExperiment-method
(mold), 25
mold,Rle-method (mold), 25
mold,RleList-method (mold), 25
mold,Seqinfo-method (mold), 25
mold,SummarizedExperiment-method
(mold), 25
mold,Views-method (mold), 25
mold,VRanges-method (mold), 25
mut.gr (CRC), 7

parseArgsForAes, 26
parseArgsForNonAes (parseArgsForAes), 26
pileupAsGRanges, 27
pileupGRangesAsVariantTable, 28
plotColorLegend, 29

showColor, 30
shrinkageFun, *24*
shrinkageFun (shrinkageFun-method), 30
shrinkageFun,GenomicRanges-method
(shrinkageFun-method), 30
shrinkageFun,IRanges-method
(shrinkageFun-method), 30
shrinkageFun-method, 30
splitByFacets, 31
splitByFacets,GRanges,formula-method
(splitByFacets), 31
splitByFacets,GRanges,GRanges-method
(splitByFacets), 31
splitByFacets,GRanges,missing-method
(splitByFacets), 31
splitByFacets,GRanges,NULL-method
(splitByFacets), 31
strip_formula_dots, 33
subsetArgsByFormals, 33

transformDfToGr (transformToGenome), 35
transformGRangesForEvenSpace, 34
transformToArch (transformToGenome), 35
transformToArch,GRanges-method
(transformToGenome), 35
transformToBarInCircle
(transformToGenome), 35
transformToCircle (transformToGenome),
35
transformToDf (transformToGenome), 35
transformToDf,GRanges-method
(transformToGenome), 35
transformToDf-method
(transformToGenome), 35
transformToGenome, 35
transformToGenome,GRanges-method
(transformToGenome), 35
transformToGenome,GRangesList-method
(transformToGenome), 35
transformToLinkInCircle
(transformToGenome), 35
transformToRectInCircle
(transformToGenome), 35
transformToSegInCircle
(transformToGenome), 35