Package 'gmoviz'

March 13, 2025

Type Package

Title Seamless visualization of complex genomic variations in GMOs and edited cell lines

Version 1.18.0

Description Genetically modified organisms (GMOs) and cell lines are widely used models in all kinds of biological research. As part of characterising these models, DNA sequencing technology and bioinformatics analyses are used systematically to study their genomes. Therefore, large volumes of data are generated and various algorithms are applied to analyse this data, which introduces a challenge on representing all findings in an informative and concise manner. `gmoviz` provides users with an easy way to visualise and facilitate the explanation of complex genomic editing events on a larger, biologically-relevant scale.

biocViews Visualization, Sequencing, GeneticVariability, GenomicVariation, Coverage

Depends circlize, GenomicRanges, graphics, R (>= 4.0)

Imports grid, gridBase, Rsamtools, ComplexHeatmap, BiocGenerics, Biostrings, GenomeInfoDb, methods, GenomicAlignments, GenomicFeatures, IRanges, rtracklayer, pracma, colorspace, S4Vectors

License GPL-3

Encoding UTF-8

LazyData true

RoxygenNote 7.0.2

Suggests testthat, knitr, rmarkdown, pasillaBamSubset, BiocStyle, BiocManager

VignetteBuilder knitr

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

git_branch RELEASE_3_20

git_last_commit 94ff6df

git_last_commit_date 2024-10-29

Repository Bioconductor 3.20

Date/Publication 2025-03-13

Author Kathleen Zeglinski [cre, aut],
Arthur Hsu [aut],
Monther Alhamdoosh [aut] (<https://orcid.org/0000-0002-2411-1325>),
Constantinos Koutsakis [aut]

Maintainer Kathleen Zeglinski <kathleen.zeglinski@csl.com.au>

Contents

`colourSets` *gmoviz colour sets*

Description

gmoviz colour sets

`nice_colours`: The default colour set. Medium brightness, light colours. Designed for use on a white/light coloured background.

`pastel_colours`:A set of pale/pastel colours, modelled on the `nice_colours` set but less saturated. Designed for use on a white/light background.

`rich_colours`: A set of bright, vibrant colours (but not neon, like the bright_colours_transparent). Designed for use on any sort of background.

`bright_colours_transparent`: A set of very bright (neon) colours with slight transparency. Designed for use on a black background.

`bright_colours_opaque`: A set of very bright (neon) colours without transparency. Designed for use on a black background.

Usage

```
nice_colours

pastel_colours

rich_colours
```

`bright_colours_transparent`

`bright_colours_opaque`

Format

Character vectors of 34 hex colours.

Details

Due to the often high number of sectors being plotted with gmoviz (e.g. 20+ when plotting each chromosome), a number of 'colour sets' have been included for convenience.

Source

Many of the colours are from, or inspired by ColorBrewer http://colorbrewer2.org/.

---

drawFeatureTrack *Add a 'feature' track to an existing plot*

Description

Adds to an existing plot a track which displays 'features' (e.g. genes, indels, primer sequences etc) using coloured shapes. Note that you must have initialised the circular plot (by `gmovizInitialise` first).

Usage

```
drawFeatureTrack(feature_data, flipped_sector = NULL,
  feature_label_cutoff = 50, track_height = 0.1,
  feature_label_size = 0.9, label_track_height = 0.1 *
  feature_label_size, coverage_rectangle = NULL, coverage_data = NULL,
  internal = FALSE, feature_outline = TRUE)
```

Arguments

`feature_data` A data frame or GRanges containing the 'features' to plot.

- GRanges input should contain `label`, `colour`, `shape` and `track` as meta-data columns.
- Data frame should contain `label`, `colour`, `shape` and `track`, as well as the additional columns `chr`, `start` and `end`

Please see below for a detailed description of these columns, and `getFeatures` for a function which can read this information in from a .gff file.

`flipped_sector` A vector of sectors that will have their genomic position (x values) reversed (ascending in anti-clockwise direction, as opposed to the usual ascending in a clock-wise direction).

`feature_label_cutoff`

To enhance readability when the shapes are small, those labels belonging to features smaller than `feature_label_cutoff` will instead be plotted on a new track closer to the centre of the circle, rather than inside the shapes themselves.

`track_height` The height (proportion of the circle) taken up by **each track** of features. The default value of `0.1` is appropriate for up to 2 feature tracks; if you get an error due to running out of space please reduce this.

`feature_label_size`
Size of the feature labels.

`label_track_height`
Size of the track on which to plot the labels.

`coverage_rectangle`, `coverage_data`
If, when initialising the graph you have used coverage_rectangleangle AND you want to plot features on the outermost track (track 0), please fill these in the same as in your `gmovizInitialise` function call. **Otherwise, there is no need to supply these**.

`internal` For internal use only.

`feature_outline`
Should a black outline be drawn around the feature shape? (It is recommended to set this to `FALSE` when dealing with very small features)

Value

Adds a 'feature' track to an existing plot.

Feature data format

The feature data GRanges contains four metadata columns:

label A character string which will be used to label the feature. It is suggested to keep this label relatively short, if possible.

colour A character string of a colour to use. Supports hex colours (*e.g. #000000*) and named R colours (*e.g. red*).

shape The shape that will be used to represent the feature:

- `'rectangle'`
- `'forward_arrow'`
- `'reverse_arrow'`
- `'upwards_triangle'` (out of the circle).
- `'downwards_triangle'` (into the circle).

It is suggested to use `'forward_arrow'` for genes on the forward strand and `'reverse_arrow'` for genes on the reverse strand.

track The index of the track on which to plot the feature:

- 0 represents the outermost track, where the ideogram rectangles that represent sequences/chromosomes are plotted.
- 1 is the conventional (default) track on which to plot a feature.
- 2, 3 and so on are further into the centre of the circle.

It is strongly recommended to keep the tracks below 3, otherwise there may not be enough space in the circle to fit them all.

These columns are all **optional**. If you don't supply them, then default values will be added as follows:

label `''`

colour a colour allocated from `rich_colours`

shape `'rectangle'`

track 1

See Also

featureDiagram for a function that, while slightly less flexible, generates an entire visualisation in one go. Also getFeatures for a function that can read the feature data in from a .gff file.

Examples

```
## plasmid map
plasmid_ideogram <- data.frame(chr='plasmid', start=0, end=2500)

plasmid_features <- GRanges(seqnames=rep('plasmid', 4),
ranges=IRanges(start=c(0, 451, 901, 1700), end=c(450, 900, 1400, 2200)),
colour = c('#d44a9f', '#4a91d4', '#7ad44a', '#d49d4a'),
label = c('promoter', 'gene', 'GFP', 'ampR'),
shape = c('rectangle', 'forward_arrow', 'forward_arrow', 'reverse_arrow'),
track = rep(1, 4))

## for a simple case like this you might as well use the featureDiagram
## function because it's only 1 function call, whereas here we need two:
gmovizInitialise(plasmid_ideogram)
drawFeatureTrack(plasmid_features)

## however the drawFeatureTrack function allows more flexibility e.g. if you
## want to add features to a plot containing numerical data for example:
## data
scatter_data <- GRanges(rep('plasmid', 50),
IRanges(start=sample(1:3000, 50), width=2),
scatter=rnorm(50, mean=4, sd=1))

## plotting
gmovizInitialise(plasmid_ideogram)
drawScatterplotTrack(plot_data=scatter_data)
drawFeatureTrack(plasmid_features, track_height = 0.15)
```

---

drawLinegraphTrack *Add a line graph track to an existing plot*

Description

Adds a line graph track to the existing plot. Must have initialised the circular plot (by gmovizInitialise first).

Usage

```
drawLinegraphTrack(plot_data, track_border_colour = "black",
  track_height = 0.3, yaxis_increment = NULL, ylim = NULL,
  line_shade_colour = "#5ab4ac", line_colour = "black",
  yaxis_label_size = 0.5, show_yaxis = TRUE, yaxis_tick_size = 0.4,
  yaxis_side = "left", yaxis_colour = "black",
  yaxis_location = CELL_META$sector.index, show_gridlines = TRUE,
  gridline_colour = "#aaaaaa")
```

Arguments

plot_data	Either: (1) a GRanges object with a metadata column of y values to plot OR (2) a data frame with four columns; chr (should match those supplied when initialising the plot); start and end (x values of the point: can both be the same if you only have a single x value for position) and then a fourth column of y values.
track_border_colour	Colour of the border of the plotting region.
track_height	The proportion (between 0 and 1) of the circle taken up by this track.
yaxis_increment	The increment the y axis and gridlines will use.
ylim	Vector of length 2; upper and lower limits for y axis.
line_shade_colour	The colour the will be used to fill in under the line. Set this to NULL if you just want the line rather than the area.
line_colour	The colour of the line itself.
yaxis_label_size	Size of the labels on the y axis.
show_yaxis	If TRUE, a y axis will be drawn.
yaxis_tick_size	Size of the ticks on the y axis.
yaxis_side	Side of the sector the y axis is on; either 'left' or 'right'.
yaxis_colour	Colour of the y axis.
yaxis_location	Sector the y axis is drawn on.
show_gridlines	If TRUE then gridlines will be drawn.
gridline_colour	Colour of the gridlines.

Value

Adds a line graph track to existing visualisation.

See Also

gmovizInitialise, which must be used to initialise the graph before this function. Also drawScatterplotTrack for a similar function which displays data as a scatterplot rather than as a line graph.

Examples

```
## you must initialise first!
small_ideo <- data.frame(chr=c('region 1', 'region 2', 'region 3'),
                         start=c(0, 0, 0), end=c(10000, 12000, 10000))
gmovizInitialise(small_ideo, custom_sector_width=c(0.3, 0.3, 0.3))

## make the data
smallplot_data <- data.frame(
chr = sample(c('region 1', 'region 2','region 3'), size=300, replace=TRUE),
start = seq(0, 10000, length.out=300), end = seq(0, 10000, length.out=300),
val = rnorm(300, 2, 0.5))
## line graph with no shading (just the line)
```

```
drawLinegraphTrack(smallplot_data, line_shade_colour=NULL)

## line graph with shading (a filled in shape)
drawLinegraphTrack(smallplot_data, line_shade_colour='#db009599')
```

`drawScatterplotTrack` *Add a scatterplot track to an existing plot*

Description

Adds a scatterplot track to the existing plot. Must have initialised the circular plot (by `gmovizInitialise` first).

Usage

```
drawScatterplotTrack(plot_data, track_border_colour = "black",
  track_height = 0.3, point_bicolour_cutoff = NULL,
  point_colour = "black", point_outline_colour = "black",
  point_size = 0.55, point_type = 21, ylim = NULL,
  yaxis_increment = NULL, show_yaxis = TRUE, yaxis_label_size = 0.6,
  yaxis_tick_size = 0.5, yaxis_location = CELL_META$sector.index,
  yaxis_side = "left", yaxis_colour = "black", show_gridlines = TRUE,
  gridline_colour = "#aaaaaa")
```

Arguments

`plot_data`	Either: (1) a GRanges object with a metadata column of y values to plot OR (2) a data frame with four columns; `chr` (should match those supplied when initialising the plot); `start` and `end` (x values of the point: can both be the same if you only have a single x value for position) and then a fourth column of y values.
`track_border_colour`	Colour of the border of the plotting region.
`track_height`	The proportion (between 0 and 1) of the circle taken up by this track.
`point_bicolour_cutoff`	A numeric threshold for the colour of the points (points above/below this number will be different colours).
`point_colour`	The fill colour of the points. If `point_bicolour_cutoff != NULL` then this should be a vector with two elements.
`point_outline_colour`	The colour of the outline of the points. If using point_bicolour_cutoff then this should be a vector with two elements.
`point_size`	Size of the points.
`point_type`	Type (shape) of the points, same as base R.
`ylim`	Vector of length 2; upper and lower limits for y axis.
`yaxis_increment`	The increment the y axis and gridlines will use.
`show_yaxis`	If `TRUE`, a y axis will be drawn.

`yaxis_label_size`
: Size of the labels on the y axis.

`yaxis_tick_size`
: Size of the ticks on the y axis.

`yaxis_location`
: Sector the y axis is drawn on.

`yaxis_side`
: Side of the sector the y axis is on; either `'left'` or `'right'`.

`yaxis_colour`
: Colour of the y axis.

`show_gridlines`
: If `TRUE` then gridlines will be drawn.

`gridline_colour`
: Colour of the gridlines.

Value

Adds a scatterplot track to existing visualisation.

See Also

`gmovizInitialise`, which must be used to initialise the graph before this function. Also `drawLinegraphTrack` for a similar function which displays data as a line graph instead.

Examples

```
## you must initialise first!
small_ideo <- data.frame(chr=c('region 1', 'region 2', 'region 3'),
                         start=c(0, 0, 0), end=c(10000, 12000, 10000))
gmovizInitialise(small_ideo, custom_sector_width=c(0.3, 0.3, 0.3))

## make the data
smallplot_data <- data.frame(
chr = sample(c('region 1', 'region 2','region 3'), size=40, replace=TRUE),
start = seq(0, 10000, length.out=40), end = seq(0, 10000, length.out=40),
val = rnorm(40, 2, 0.5))

## scatterplot where all points are the same colour
drawScatterplotTrack(smallplot_data)

## scatterplot with bi-colour cutoff of 2
drawScatterplotTrack(smallplot_data, point_bicolour_cutoff=2,
                     point_colour=c('red', 'blue'),
                     point_outline_colour=c('black', 'black'))
```

---

featureDiagram *Display 'features' of interest in a diagram*

Description

Generates a diagram which displays 'features' (e.g. genes, indels, primer sequences etc) using coloured shapes. See `insertionDiagram` for a similar function which specialises in plotting insertions or `drawFeatureTrack` to add a feature track to an existing graph.

Usage

```
featureDiagram(ideogram_data, feature_data, start_degree = 180,
  coverage_rectangle = NULL, coverage_data = NULL,
  custom_sector_width = NULL, space_between_sectors = 4,
  flipped_sector = NULL, sector_colours = nice_colours,
  sector_border_colours = nice_colours, sector_labels = TRUE,
  sector_label_size = 1.3, sector_label_colour = "black",
  label_data = NULL, label_size = 1.1, label_colour = "black",
  xaxis = TRUE, xaxis_label_size = 0.9, xaxis_colour = "#747577",
  xaxis_spacing = 10, feature_label_cutoff = 50,
  xaxis_spacing_unit = "deg", track_height = 0.1,
  feature_label_size = 0.9, link_data = NULL,
  link_colour = "#84c6d6", link_ends = "default", custom_ylim = NULL,
  label_track_height = 0.1 * feature_label_size,
  feature_outline = TRUE)
```

Arguments

`ideogram_data` Either a GRanges representing regions of interest or a data frame in bed format (containing the `chr`, `start` and `end` columns). If you want to read in data from file, please see the `getIdeogramData` function.

`feature_data` A data frame or GRanges containing the 'features' to plot.

- GRanges input should contain `label`, `colour`, `shape` and `track` as meta-data columns.
- Data frame should contain `label`, `colour`, `shape` and `track`, as well as the additional columns `chr`, `start` and `end`

Please see below for a detailed description of these columns, and `getFeatures` for a function which can read this information in from a .gff file.

`start_degree` Where on the circle the first sector will start being drawn from (90 = 12 o'clock).

`coverage_rectangle`
A vector containing the name(s) of any sector(s) that you would like to depict as 'coverage rectangles': filled in shapes that are a plot of the coverage data over that sector. See the example below or the vignette for an example of this.

`coverage_data` A GRanges (or data frame) containing the coverage data to plot for those sectors in `coverage_rectangle`. To read this data in from a BAM file, please see the `getCoverage` function.

`custom_sector_width`
Normally, the size of each sector is proportional to its relative length, but `custom_sector_width` can change this. It is a vector of sector sizes (as proportions of the entire circle), given in the same order in which sectors are plotted: firstly 'chr1', 'chr2' ... through to 'chrX' and 'chrY' followed by any differently named sectors e.g. 'gene 1', plasmid' in alphabetical order.

`space_between_sectors`
Space between each sector.

`flipped_sector` A vector of sectors that will have their genomic position (x values) reversed (ascending in anti-clockwise direction, as opposed to the usual ascending in a clock-wise direction).

`sector_colours` Either a single colour (which will be applied to all sectors) or a vector with the same length as the number of sectors/regions. This package includes 5 colour

sets: nice_colours, pastel_colours, bright_colours_transparent, bright_colours_opaque and rich_colours. See colourSets for more information about these.

sector_border_colours
: Same as sector_colours, only for the border of each sector.

sector_labels
: If TRUE, labels ('chr1', 'chr2' etc.) will be drawn for each sector (recommended).

sector_label_size
: Size of the sector labels.

sector_label_colour
: Colour of the sector labels.

label_data
: Data frame or GRanges containing the labels. If a GRanges, label should be a metadata column containing the character strings of the labels. type and colour can also be used to store additional information about the type (e.g. 'gene' or 'promoter') and colour of the label. This information can be used to **colour code** the labels by supplying the colour column as the label_colour parameter. Data frames should additionally include the chr, start, end which dictate the position of the label.

label_size
: Size of the labels.

label_colour
: Colour of the labels, can be either a single value (applied to all labels) or a vector with the same length as the number of labels (for colour-coding).

xaxis
: If TRUE, an x (genomic position) xaxis will be plotted.

xaxis_label_size
: Size of the x axis labels.

xaxis_colour
: Colour of the x axis labels.

xaxis_spacing
: Space between the x axis labels, in degrees. Alternatively, the string 'start_end' will place a label at the start and end of each sector only.

feature_label_cutoff
: To enhance readability when the shapes are small, those labels belonging to features smaller than feature_label_cutoff will instead be plotted on a new track closer to the centre of the circle, rather than inside the shapes themselves.

xaxis_spacing_unit
: Either "deg" to draw ticks every certain number of degrees around the circle or "bp" to draw ticks every certain bp around the circle (be warned that when the scales for each sector are very different, it's best to use "deg")

track_height
: The height (vertical distance around the circle) that will be taken up by this track. Should be a number between 0 (none) and 1 (entire circle).

feature_label_size
: Size of the feature labels.

link_data
: If you would like to draw a link between two sectors of the circle, link_data should be a data frame with two rows: one for each end of the link. There should be 3 columns: chr, start & end which describe the position of each end of the link.

link_colour
: The colour of the link: this should be a 6 digit hex code, the transparency is automatically added.

link_ends
: How far the link extends in either direction. *This is set automatically* but if you want to edit it, provide a vector of length 2 with each element being between 0 (centre of circle) and 1 (right at the edge of the circle).

custom_ylim
: A vector of length two containing the y (coverage) axis limits. No need to set if not using coverage rectangles or if you're happy with the default: c(0, maximum coverage).

`label_track_height`
: Size of the track on which to plot the labels.

`feature_outline`
: Should a black outline be drawn around the feature shape? (It is recommended to set this to `FALSE` when dealing with very small features)

Value

Generates an image of the feature data supplied.

Warning

If you choose to use a data frame to supply the feature data, please be careful to add the `stringsAsFactors = FALSE` argument. Otherwise, the colours may not be correct.

See Also

`insertionDiagram` for a more specialised function which shows the copy number of insertions. Also `drawFeatureTrack` to add the exact same feature information to an existing plot and `getFeatures` for a function that can read in the feature information from a .gff file.

Examples

```
## plasmid map
plasmid_ideogram <- data.frame(chr='plasmid', start=0, end=2500)

plasmid_features <- GRanges(seqnames=rep('plasmid', 4),
ranges=IRanges(start=c(0, 451, 901, 1700), end=c(450, 900, 1400, 2200)),
colour=c('#d44a9f', '#4a91d4', '#7ad44a', '#d49d4a'),
label=c('promoter', 'gene', 'GFP', 'ampR'),
shape=c('rectangle', 'forward_arrow', 'forward_arrow', 'reverse_arrow'),
track=rep(1, 4))

featureDiagram(plasmid_ideogram, plasmid_features)
```

---

getCoverage *Import coverage data from .bam file*

Description

Uses RSamtools to import coverage data from .bam file and format it appropriately for plotting with gmoviz.

Usage

```
getCoverage(regions_of_interest, bam_file, window_size = 1,
  smoothing_window_size = NULL)
```

Arguments

`regions_of_interest`
: either a GRanges of regions OR a character vector of sequences/chromosomes to find the coverage for (please be careful that the names here match the spelling/format of those in the bam file).

`bam_file`
: Location of the bam file from which to read coverage data.

`window_size`
: The size of the window to for calculating coverage (default is 1; per base coverage). Use `smoothCoverage` to smooth the data, this is more for reducing time taken to read in and plot coverage over a large number of bases.

`smoothing_window_size`
: If supplied, then moving average smoothing will be applied using the `movavg` function from the package **pracma** (please make sure it's installed). Note: smoothing may take some time when there are many points involved. Please be patient, or proceed without smoothing.

Value

A GRanges containing the coverage data in the metadata column 'coverage'.

See Also

The `gmovizInitialise`, `drawLinegraphTrack`, `insertionDiagram` and `featureDiagram` functions which can plot the coverage data.

Examples

```
## the example .bam file
path <- system.file('extdata', 'ex1.bam', package='Rsamtools')

## example without smoothing or windowing
getCoverage(regions_of_interest='seq1', bam_file=path)

## using windowing
getCoverage(regions_of_interest='seq1', bam_file=path, window_size=5)

## using smoothing
getCoverage(regions_of_interest='seq1', bam_file=path,
smoothing_window_size=3)

## specifying only a particular region to read in using GRanges
small_range <- GRanges('seq1', IRanges(0, 500))
getCoverage(regions_of_interest=small_range, bam_file=path)
## please be very careful that the sequence names are spelled exactly like
## in the bam file or you'll get an error! The following WON'T WORK.
## Not run:
getCoverage(regions_of_interest='chr1', bam_file=path)
## End(Not run)
```

getFeatures *Generate a GRanges containing 'features' from .gff files*

Description

Uses a .gff file to create a GRanges of 'features' (e.g. genes or other regions of interest within the genome) which can then be plotted with the `featureDiagram` or `drawFeatureTrack` functions.

Usage

```
getFeatures(gff_file, colours = nice_colours, colour_by_type = TRUE)
```

Arguments

`gff_file`	Location of the gff file to read in.
`colours`	A character vector of colours to be used to colour code the features.
`colour_by_type`	If `TRUE`, the features will be coloured according to the 'type' field of the gff file. If `FALSE`, colours will be assigned based on the name of the feature (each uniquely named feature gets its own colour).

Value

A GRanges containing the 'features'. See `drawFeatureTrack` for a detailed description of the format.

See Also

`getLabels` for a function which reads the entries of a .gff file into labels rather than 'features'. Also `featureDiagram` or `drawFeatureTrack` for functions which can plot this data.

Examples

```
## the example .gff
path <- system.file('extdata', 'example.gff3', package='gmoviz')

## coloured by type
getFeatures(path)

## not coloured by type (each uniquely named feature gets its own colour)
getFeatures(path, colour_by_type=FALSE)
```

`getIdeogramData` *Import transgenic genome data from .bam or .fasta file*

Description

Read in the seqname, start & end from .bam or .fasta file and format correctly for plotting with gmoviz.

Usage

```
getIdeogramData(bam_file = NULL, fasta_file = NULL,
  fasta_folder = NULL, just_pattern = NULL, unwanted_chr = NULL,
  wanted_chr = NULL, add_chr = TRUE)
```

Arguments

`bam_file`, `fasta_file`, `fasta_folder`	Location of either a .bam file, .fasta file or folder of .fasta files to read in. You only need to supply one of these file types; .bam files are recommended because it is much faster than using .fasta files. Also note that the filters `unwatedChr`, `wanted_chr` and `just_pattern` won't work with single .fasta files (only with .bam or .fasta folders).
`just_pattern`	If supplied, this pattern (regex) will be used to select the sequences to read in
`unwanted_chr`	If supplied, these sequences won't be read in
`wanted_chr`	If supplied, only these sequences will be read in
`add_chr`	If `TRUE`, 'chr' will be added to the start of sequence names with one or two characters (e.g. X will become chrX and 10 will become chr10 but plasmid will remain as-is)

Value

A GRanges containing the ideogram data (sequence names, starts & ends).

See Also

The `gmovizInitialise` and `featureDiagram` functions which can be used to plot this data.

Examples

```
## the example .bam file
path <- system.file('extdata', 'ex1.bam', package='Rsamtools')

## just starting with 'seq'
getIdeogramData(bam_file=path, just_pattern='^seq')

## only seq1
getIdeogramData(bam_file=path, wanted_chr='seq1')

## not seq2 (same as above)
getIdeogramData(bam_file=path, unwanted_chr='seq2')
```

```
## you can mix and match any of the filters
getIdeogramData(bam_file=path, unwanted_chr='seq2', just_pattern='^seq')

## the function also works to read in individual .fasta files, but please
## note that for now the filters won't work (so if you have multiple
## sequences in one .fasta file then they will all be read in)
path <- system.file('extdata', 'someORF.fa', package='Biostrings')
getIdeogramData(fasta_file=path)

## we can also read in selected .fasta files from a folder of .fasta files,
## based on the filters shown above for the .bam file
path <- system.file('extdata', 'fastaFolder', package='gmoviz')
getIdeogramData(fasta_folder=path)
```

`getLabels` *Generate a GRanges of labels from .gff files*

Description

Uses a .gff file to create a GRanges of labels which can then be plotted with the `label_data` argument of many functions in this package such as `gmovizInitialise`, `insertionDiagram` or `featureDiagram`.

Usage

```
getLabels(gff_file, colour_code = TRUE,
  colours = bright_colours_opaque)
```

Arguments

`gff_file`	Location of the gff file to read in.
`colour_code`	If `TRUE`, the labels will be assigned colours according to the 'type' field of the gff file. If `FALSE`, colours will not be assigned.
`colours`	A character vector of colours to be used to colour code the labels (if `colour_code` is `TRUE`).

Value

A GRanges containing the gene label data. See `gmovizInitialise` for a detailed description of the format.

See Also

`getFeatures` for a function which reads the entries of a .gff file into 'features' rather than labels. Also `gmovizInitialise`, `insertionDiagram` and `featureDiagram` for functions which can plot this data.

Examples

```
## example .gff
path <- system.file('extdata', 'example.gff3', package='gmoviz')

## colour coded
getLabels(path)

## not colour coded (all black)
getLabels(path, colour_code=FALSE)
```

---

gmovizInitialise *Initialise the layout of the circular plot*

Description

Draws the ideogram (sectors around a circle representing sequences of interest, like chromosomes), labels and genomic axis in preparation for the addition of other tracks like drawFeatureTrack or drawLinegraphTrack.

Usage

```
gmovizInitialise(ideogram_data, start_degree = 90,
  space_between_sectors = 1, zoom_sectors = NULL, zoom_size = 0.055,
  remove_unzoomed = TRUE, zoom_prefix = "zoomed_",
  custom_sector_width = NULL, track_height = 0.1,
  sector_colours = nice_colours, sector_border_colours = nice_colours,
  coverage_rectangle = NULL, coverage_data = NULL,
  custom_ylim = NULL, sector_labels = TRUE, sector_label_size = 0.9,
  sector_label_colour = "black", xaxis = TRUE,
  xaxis_orientation = "top", xaxis_label_size = 0.75,
  xaxis_colour = "#747577", xaxis_spacing = 10,
  xaxis_spacing_unit = "deg", label_data = NULL,
  label_colour = "black", label_size = 0.85,
  space_between_labels = 0.4, label_orientation = "outside",
  sort_sectors = TRUE)
```

Arguments

ideogram_data	Either a GRanges representing regions of interest or a data frame in bed format (containing the chr, start and end columns). If you want to read in data from file, please see the getIdeogramData function.
start_degree	Where on the circle the first sector will start being drawn from (90 = 12 o'clock).
space_between_sectors	Space between each sector.
zoom_sectors	A character vector of sectors that should be 'zoomed' (made bigger than usual, useful to show shorter sequences like plasmids alongside longer sequences like chromosomes).
zoom_size	The size of the zoomed chromosome, as a proportion of the entire circle (0 = invisible, 1 = entire circle filled). The default value of 0.055 is good for displaying something small (e.g. plasmid) alongside something large (e.g. chromosomes).

remove_unzoomed
: If TRUE, the sectors in zoom_sectors will only appear in their zoomed form. If FALSE, both the zoomed and unzoomed versions will be plotted.

zoom_prefix
: A character prefix that will be applied to zoomed sequences to distinguish them from non-zoomed ones.

custom_sector_width
: Normally, the size of each sector is proportional to its relative length, but custom_sector_width can change this. It is a vector of sector sizes (as proportions of the entire circle), given in the same order in which sectors are plotted: firstly 'chr1', 'chr2' ... through to 'chrX' and 'chrY' followed by any differently named sectors e.g. 'gene 1', plasmid' in alphabetical order.

track_height
: The height (vertical distance around the circle) that will be taken up by this track. Should be a number between 0 (none) and 1 (entire circle).

sector_colours
: Either a single colour (which will be applied to all sectors) or a vector with the same length as the number of sectors/regions. This package includes 5 colour sets: nice_colours, pastel_colours, bright_colours_transparent, bright_colours_opaque and rich_colours. See colourSets for more information about these.

sector_border_colours
: Same as sector_colours, only for the border of each sector.

coverage_rectangle
: A vector containing the name(s) of any sector(s) that you would like to depict as 'coverage rectangles': filled in shapes that are a plot of the coverage data over that sector. See the example below or the vignette for an example of this.

coverage_data
: A GRanges (or data frame) containing the coverage data to plot for those sectors in coverage_rectangle. To read this data in from a BAM file, please see the getCoverage function.

custom_ylim
: A vector of length two containing the y (coverage) axis limits. No need to set if not using coverage rectangles or if you're happy with the default: c(0, maximum coverage).

sector_labels
: If TRUE, labels ('chr1', 'chr2' etc.) will be drawn for each sector (recommended).

sector_label_size
: Size of the sector labels.

sector_label_colour
: Colour of the sector labels.

xaxis
: If TRUE, an x (genomic position) xaxis will be plotted.

xaxis_orientation
: Either 'top' to put the x axis on the outside of the circle or 'bottom' to put it on the inside.

xaxis_label_size
: Size of the x axis labels.

xaxis_colour
: Colour of the x axis labels.

xaxis_spacing
: Space between the x axis labels, in degrees. Alternatively, the string 'start_end' will place a label at the start and end of each sector only.

xaxis_spacing_unit
: Either "deg" to draw ticks every certain number of degrees around the circle or "bp" to draw ticks every certain bp around the circle (be warned that when the scales for each sector are very different, it's best to use "deg")

label_data — Data frame or GRanges containing the labels. If a GRanges, label should be a metadata column containing the character strings of the labels. type and colour can also be used to store additional information about the type (e.g. 'gene' or 'promoter') and colour of the label. This information can be used to **colour code** the labels by supplying the colour column as the label_colour parameter. Data frames should additionally include the chr, start, end which dictate the position of the label.

label_colour — Colour of the labels, can be either a single value (applied to all labels) or a vector with the same length as the number of labels (for colour-coding).

label_size — Size of the labels.

space_between_labels — Space between the labels

label_orientation — 'outside' to put the labels on the outside of the circle, 'inside' to put them on the inside.

sort_sectors — If TRUE, the sectors will be plotted around the circle in alphabetical order. Otherwise, they will be in the order in which they appear in ideogram_data

Value

Generates an image of the initial ideogram track which can then be added to with various other functions.

See Also

The drawScatterplotTrack, drawFeatureTrack and drawLinegraphTrack, which can be used to add information to this plot. Also getIdeogramData which can be used to read in the needed ideogram data for this function.

Examples

```
## normal/standard usage
ideogram <- data.frame(chr=paste0('chr', c(1:19, 'X', 'Y')),
start=rep(0, 21),
end=c(195471971, 182113224, 160039680, 156508116, 151834684, 149736546,
145441459, 129401213, 124595110, 130694993, 122082543, 120129022,
120421639, 124902244, 104043685, 98207768, 94987271, 90702639, 61431566,
171031299, 91744698))
gmovizInitialise(ideogram)

## zooming a sector
gmovizInitialise(ideogram, zoom_sectors='chr19', zoom_size=0.2)

## custom sector width
small_ideogram <- data.frame(chr=c('region 1', 'region 2', 'region 3'),
start=c(0, 0, 0), end=c(10000, 12000, 10000))
gmovizInitialise(small_ideogram, custom_sector_width=c(0.3, 0.3, 0.3))

## coverage rectangle
path <- system.file('extdata', 'ex1.bam', package='Rsamtools')
ideo <- getIdeogramData(path, wanted_chr='seq1')
coverage <- getCoverage(bam_file=path, regions_of_interest='seq1',
window_size=30)
gmovizInitialise(ideo, coverage_rectangle='seq1', coverage_data=coverage)
```

gmovizPlot *Generate an entire circular plot*

Description

Saves code supplied to `plotting_functions` a plot (with optional title and legends) as either .png, .svg or .ps.

Usage

```
gmovizPlot(file_name, file_type = "png", plotting_functions,
  legends = NULL, title = NULL, width = 338.7, height = 238.7,
  units = "mm", res = 300, background_colour = "transparent",
  title_x_position = 0.5, title_y_position = 0.9,
  title_font_face = "bold", title_size = 1.1, title_colour = "black",
  point_size = 11)
```

Arguments

`file_name`	The name of the file to be saved.
`file_type`	The type of image file to produce: either `'png'`, `'svg'` or `'ps'`.
`plotting_functions`	The functions you want to plot (e.g. `insertionDiagram` or `gmovizInitialise`).
`legends`	A legend object to plot, generated by `makeLegends`.
`title`	Text for the title, leave as `NULL` for no title.
`width`	Width of the image.
`height`	Height of the image.
`units`	Units for the width and height of the image. One of `'mm'`, `'cm'` or `'in'` (inches).
`res`	Resolution of the image (only needed for .png files).
`background_colour`	Colour of the image background.
`title_x_position, title_y_position`	X and Y positions of the title on the image.
`title_font_face`	Font face of the title: bold, italic or bold-italic.
`title_size`	Size of the title.
`title_colour`	Colour of the title.
`point_size`	Pointsize (for postscript output only).

Value

Saves a plot to disk in the specified format.

See Also

`makeLegends` for a function that generates the legend objects.

Examples

```
## make some example data
small_ideogram <- data.frame(chr=c('region 1', 'region 2', 'region 3'),
start=c(0, 0, 0), end=c(10000, 12000, 10000))
small_plot_data <- data.frame(
chr=sample(c('region 1', 'region 2', 'region 3'), size=40, replace=TRUE),
start=sample(0:10000, 40), end=sample(0:10000, 40),
val=rnorm(40, 2, 0.5))

## plot it
## Not run:
gmovizPlot('test.png', {
gmovizInitialise(small_ideogram, custom_sector_width=c(0.3, 0.3, 0.3))
drawScatterplotTrack(small_plot_data)}, title='scatterplot')
## End(Not run)
```

`insertionDiagram` *Display number of copies of an insertion*

Description

Generates a diagram which displays insertions, showing their position, size and copy number. See `featureDiagram` for a more general function which can display other features of interest.

Usage

```
insertionDiagram(insertion_data, style = 1, either_side = "default",
  insertion_label = "default", sector_colours = nice_colours,
  sector_border_colours = nice_colours, start_degree = 180,
  custom_sector_width = NULL, coverage_rectangle = NULL,
  coverage_data = NULL, custom_ylim = NULL,
  space_between_sectors = 15, sector_labels = TRUE,
  sector_label_size = 1.3, sector_label_colour = "black",
  label_data = NULL, label_colour = "black", link_colour = "default",
  label_size = 1.1, xaxis = TRUE, xaxis_label_size = 0.9,
  xaxis_colour = "#747577", xaxis_spacing = 10,
  xaxis_spacing_unit = "deg", link_ends = "default",
  track_height = 0.15, internal = FALSE)
```

Arguments

`insertion_data`	A `GRanges` or data frame describing the insertion. See below for the detailed format.
`style`	How the original sequence and insertions will be positioned around the circle. Choose from options 1, 2, 3 or 4. Please see the examples below or the vignette for what these options represent.
`either_side`	How much extra of the genome should be shown around the insertion site. Can be either a single number (e.g. `1000`, then 1000bp will be shown either side of the insertion site), a vector of length 2 (e.g. `c(2000, 13000)` in which case from 2000 to 13000 will be shown) or a GRanges (in which case all ranges in the GRanges object will be used to determine the start/end points of the sector)

`insertion_label`
: The label(s) that will be applied to the insertions. If `'default'` then the name of the insertion will be used to label single copy insertions and a number will be used for multiple copy number insertions. Otherwise, `insertion_label` should be a vector with one element for each row of the insertion data, indicating the label that should be used for that insertion.

`sector_colours`
: Either a single colour (which will be applied to all sectors) or a vector with the same length as the number of sectors/regions. This package includes 5 colour sets: `nice_colours`, `pastel_colours`, `bright_colours_transparent`, `bright_colours_opaque` and `rich_colours`. See `colourSets` for more information about these.

`sector_border_colours`
: Same as `sector_colours`, only for the border of each sector.

`start_degree`
: Where on the circle the first sector will start being drawn from (90 = 12 o'clock).

`custom_sector_width`
: Normally, the size of each sector is proportional to its relative length, but `custom_sector_width` can change this. It is a vector of sector sizes (as proportions of the entire circle), given in the same order in which sectors are plotted: firstly 'chr1', 'chr2' ... through to 'chrX' and 'chrY' followed by any differently named sectors e.g. 'gene 1', plasmid' in alphabetical order.

`coverage_rectangle`
: A vector containing the name(s) of any sector(s) that you would like to depict as 'coverage rectangles': filled in shapes that are a plot of the coverage data over that sector. See the example below or the vignette for an example of this.

`coverage_data`
: A GRanges (or data frame) containing the coverage data to plot for those sectors in `coverage_rectangle`. To read this data in from a BAM file, please see the `getCoverage` function.

`custom_ylim`
: A vector of length two containing the y (coverage) axis limits. No need to set if not using coverage rectangles or if you're happy with the default: c(0, maximum coverage).

`space_between_sectors`
: Space between each sector.

`sector_labels`
: If `TRUE`, labels ('chr1', 'chr2' etc.) will be drawn for each sector (recommended).

`sector_label_size`
: Size of the sector labels.

`sector_label_colour`
: Colour of the sector labels.

`label_data`
: Data frame or GRanges containing the labels. If a GRanges, `label` should be a metadata column containing the character strings of the labels. `type` and `colour` can also be used to store additional information about the type (e.g. 'gene' or 'promoter') and colour of the label. This information can be used to **colour code** the labels by supplying the colour column as the `label_colour` parameter. Data frames should additionally include the `chr`, `start`, `end` which dictate the position of the label.

`label_colour`
: Colour of the labels, can be either a single value (applied to all labels) or a vector with the same length as the number of labels (for colour-coding).

`link_colour`
: The colour of the link: this should be a 6 digit hex code, the transparency is automatically added.

`label_size`
: Size of the labels.

`xaxis`
: If `TRUE`, an x (genomic position) xaxis will be plotted.

`xaxis_label_size`
: Size of the x axis labels.

`xaxis_colour`
: Colour of the x axis labels.

`xaxis_spacing`
: Space between the x axis labels, in degrees. Alternatively, the string 'start_end' will place a label at the start and end of each sector only.

`xaxis_spacing_unit`
: Either `"deg"` to draw ticks every certain number of degrees around the circle or `"bp"` to draw ticks every certain bp around the circle (be warned that when the scales for each sector are very different, it's best to use `"deg"`)

`link_ends`
: How far the link extends in either direction. *This is set automatically* but if you want to edit it, provide a vector of length 2 with each element being between 0 (centre of circle) and 1 (right at the edge of the circle).

`track_height`
: The height (vertical distance around the circle) that will be taken up by this track. Should be a number between 0 (none) and 1 (entire circle).

`internal`
: For internal use only.

Value

Generates an image displaying the copy number of the insertion(s) provided

Insertion data format

The start, end and seqnames of insertion_data GRanges should describe the insertion site. Additionally, there are five metadata columns:

name A character string which will be used to label insertion. It is suggested to keep this label relatively short, if possible.

colour A character string of a colour to use. Supports hex colours (*e.g. #000000*) and named R colours (*e.g. red*).

shape The shape that will be used to represent the feature:

- `'rectangle'` is a rectangle.
- `'forward_arrow'` for a forwards facing arrow.
- `'reverse_arrow'` for a backwards (reverse) facing arrow.

It is suggested to use `'forward_arrow'`

length The length of the insertion

in_tandem The number of copies of the insert in tandem

The columns **in_tandem, colour and shape are all optional**. If you don't supply them, then default values will be added as follows:

in_tandem 1 (only one copy inserted)

colour a colour allocated from `rich_colours`

shape `'forward_arrow'`

Warning

If you choose to use a data frame to supply the insertion_data, please be careful to add the `stringsAsFactors=FALSE` argument. Otherwise, the colours may not be correct.

See Also

featureDiagram for a more flexible function that takes a similar approach to representing features of interest.

Examples

```
## one insertion with 4 tandem copies
## the data as a data.frame
exampleins <- data.frame(
chr='chr12', start=70905597, end=70917885, name='plasmid',
colour='#7270ea', length=12000, in_tandem=11, shape='forward_arrow',
stringsAsFactors=FALSE)

## or we can supply it as GRanges (same thing)
exampleins <- GRanges(
seqnames='chr12', ranges=IRanges(start=70905597, end=70917885),
name='plasmid', colour='#7270ea', length=12000, in_tandem=11,
shape='forward_arrow')

## plot it
insertionDiagram(exampleins, either_side=c(70855503, 71398284))

## that was the default 'style'. The other 3 styles are:
## style 2
insertionDiagram(exampleins, either_side=c(70855503, 71398284), style=2)

## style 3
insertionDiagram(exampleins, either_side=c(70855503, 71398284), style=3)

## style 4
insertionDiagram(exampleins, either_side=c(70855503, 71398284), style=4)

## 2 different insertions
## the data
example2ins <- data.frame(
chr=c('chr12', 'chr12'), start=c(70905597, 70705597),
end=c(70917885, 70717885), name=c('plasmid1', 'plasmid2'),
colour=c('#7270ea', '#ea7082'), length=c(12000, 10000),
in_tandem=c(4, 8), shape=c('reverse_arrow', 'forward_arrow'),
stringsAsFactors=FALSE)

## plot it
insertionDiagram(example2ins, link_colour='#ffe677', start_degree=45)
```

---

makeLegends *Add a legend*

Description

Makes a legend object using ComplexHeatmap package which can then be plotted using the gmovizPlot function.

Usage

```
makeLegends(label_legend = FALSE, label_data = NULL,
  label_legend_title = "Gene Labels", feature_legend = FALSE,
  feature_data = NULL, feature_legend_title = "Features",
  scatterplot_legend = FALSE, scatterplot_legend_labels = c("Gains",
  "Losses"), point_colour = "black", point_outline_colour = "black",
  point_type = 21, scatterplot_legend_title = "Copy Number Variants",
  linegraph_legend = FALSE,
  linegraph_legend_labels = "Per Base Coverage",
  linegraph_legend_colours = "black",
  linegraph_legend_title = "Line Graph", background_colour = "white")
```

Arguments

`label_legend`	Whether to make a legend for labels (good for colour-coded labels).
`label_data`	The label data.
`label_legend_title`	Title for the label legend.
`feature_legend`	Whether to make a legend for features.
`feature_data`	The feature data to use for the feature legend.
`feature_legend_title`	Title for the features legend.
`scatterplot_legend`	Whether to make a legend for the scatterplot track.
`scatterplot_legend_labels`	A vector of the name/description of each point e.g. if a point represents methylation, use 'methylation'. If we have red/blue points for copy number gain/loss use c('gain', 'loss').
`point_type`, `point_colour`, `point_outline_colour`	The type and colour of points, as supplied to the `drawScatterplotTrack` function.
`scatterplot_legend_title`	Title for scatterplot track legend.
`linegraph_legend`	Whether to plot a legend for a line graph track.
`linegraph_legend_labels`	A vector of label(s) for what the line graph means (e.g. `'Per Base Coverage'` for a line graph track showing coverage).
`linegraph_legend_colours`	The colour of to the line graph track.
`linegraph_legend_title`	A title for the line graph legend.
`background_colour`	The colour of the background (either 'white' or 'black').

Value

An object of the Legends class.

See Also

If you want more customisation over your legends, please see https://jokergoo.github.io/circlize_book/book/legends.html for a detailed guide as to how to implement legends alongside the circlize plots. To plot these legends, see gmovizPlot

Examples

```
## a gene label legend
## the data
labels <- data.frame(chr=c('chr1', 'chr1'), start=c(100, 300),
end=c(150, 350), label=c('a', 'b'), type=c('gene', 'lncRNA'),
colour=c('red', 'blue'))

## making the legend
makeLegends(label_legend=TRUE, label_data=labels)
```

---

multipleInsertionDiagram

Display multiple insertion events around a genome

Description

Generates a diagram which displays multiple insertion events (as displayed using the insertionDiagram function) around a central genome

Usage

```
multipleInsertionDiagram(insertion_data, genome_ideogram_data,
  either_side = "default", track_height = 0.15, style = 1,
  colour_set = nice_colours, coverage_rectangle = NULL,
  coverage_data = NULL, label_data = NULL, label_colour = "black",
  label_size = 1, xaxis_spacing = "start_end")
```

Arguments

insertion_data	A GRanges or data frame describing each of the insertion events. Please see insertionDiagram for a detailed description of the format.
genome_ideogram_data	Either a GRanges representing regions of interest or a data frame in bed format (containing the chr, start and end columns). If you want to read in data from file, please see the getIdeogramData function.
either_side	How much extra of the genome should be shown around the insertion site. See insertionDiagram for a description of the different ways you can specify either_side, but note that for this function you need to supply either one value (which will apply to all of the events) or a named list of values (with one element per event. The names should be the names of the insertion _NOT_ the names of the chromosomes).
track_height	The height (vertical distance around the circle) that will be taken up by this track. Should be a number between 0 (none) and 1 (entire circle) that will apply to all of the events.

`style` How the original sequence and insertions are positioned around the circle (style 1, 2, 3 or 4). Please see the examples of the `insertionDiagram` function or the vignette for what each of these options look like. This should be either a single value (which will apply to all of the events) or a named vector of values (with one element per event).

`colour_set` The set of colours that will be used to create the diagram. For simplicity, it isn't possible to specify precisely the colour of each sector and link in the diagram (but you can easily edit them by saving the diagram in a vectorised format and opening it in any vector graphics editing program). See `colourSets` for the built-in gmoviz colour sets or make your own (should be a vector of hex colours; must have a length greater than or equal to the number of rows of genome_ideogram_data)

`coverage_rectangle` A vector containing the name(s) of any sector(s) that you would like to depict as 'coverage rectangles': filled in shapes that are a plot of the coverage data over that sector. See the example below or the vignette for an example of this.

`coverage_data` A GRanges (or data frame) containing the coverage data to plot for those sectors in `coverage_rectangle`. To read this data in from a BAM file, please see the `getCoverage` function.

`label_data` Data frame or GRanges containing the labels. If a GRanges, `label` should be a metadata column containing the character strings of the labels. `type` and `colour` can also be used to store additional information about the type (e.g. 'gene' or 'promoter') and colour of the label. This information can be used to **colour code** the labels by supplying the colour column as the `label_colour` parameter. Data frames should additionally include the `chr`, `start`, `end` which dictate the position of the label.

`label_colour` Colour of the labels, can be either a single value (applied to all labels) or a vector with the same length as the number of labels (for colour-coding).

`label_size` Size of the labels.

`xaxis_spacing` Space between the x axis labels, in degrees. Alternatively, the string 'start_end' will place a label at the start and end of each sector only. Accepts only a single value which will be applied to all events.

Value

Generates an image of the multiple insertion events provided.

Warning

Due to space limitations, it isn't possible to display more than 8 events or more than 3 events in the same quarter of the circle. If you have more events than this, please consider splitting them across two or more figures.

See Also

`insertionDiagram` for a function which generates the figures for each of the individual events and `gmovizInitialise` for the function which draws the central genome

Examples

```
## the data
ideogram_data <- GRanges(
seqnames=paste0('chr', 1:6), ranges=IRanges(start=rep(0, 6),
 end=rep(12000, 6)))
insertion_data <- GRanges(
seqnames = c('chr1', 'chr5'),
ranges = IRanges(start = c(4000, 2000), end = c(4100, 2200)),
name = c('ins1', 'ins5'), length = c(100, 200))

## the plot
multipleInsertionDiagram(insertion_data=insertion_data,
                         genome_ideogram_data=ideogram_data)
## with coverage and labels
example_labels <- GRanges(seqnames=c('chr1', 'chr5'),
                          ranges=IRanges(start=c(4000, 2000),
                          end=c(4120, 2200)),
                          label=c('Gene A', 'Gene B'),
                          colour=c('red', 'blue'))

example_coverage <- GRanges(
seqnames = c(rep('chr1', 100), rep('chr5', 100)),
ranges = IRanges(start=c(seq(4000, 4099, length.out=100),
                         seq(2000, 2199, length.out=100)),
                 end=c(seq(4001, 4100, length.out=100),
                       seq(2001, 2200, length.out=100))),
                 coverage=c(runif(100, 0, 25), runif(100, 0, 15)))
multipleInsertionDiagram(insertion_data=insertion_data,
                         genome_ideogram_data=ideogram_data,
                         label_data=example_labels,
                         label_colour=example_labels$colour,
                         coverage_rectangle=c('chr1', 'chr5'),
                         coverage_data=example_coverage)
## changing either_side and style
either_side_GRange <- GRanges('chr5', IRanges(1000, 3200))
multipleInsertionDiagram(insertion_data=insertion_data,
                         genome_ideogram_data=ideogram_data,
                         style=c('ins1'=1, 'ins5'=4),
                         either_side=list('ins1'=500,
                                          'ins5'=either_side_GRange))
```

Index