Package ‘circlize’

May 1, 2020

Type Package

Title Circular Visualization

Version 0.4.9

Date 2020-04-28

Author Zuguang Gu

Maintainer Zuguang Gu <z.gu@dkfz.de>

Depends R (>= 3.0.0), graphics

Imports GlobalOptions (>= 0.1.0), shape, grDevices, utils, stats,
colorspace, methods, grid

Suggests knitr, dendextend (>= 1.0.1), ComplexHeatmap (>= 2.0.0),
gридBase, png

VignetteBuilder knitr

Description Circular layout is an efficient way for the visualization of huge amounts of information. Here this package provides an implementation of circular layout generation in R as well as an enhancement of available software. The flexibility of the package is based on the usage of low-level graphics functions such that self-defined high-level graphics can be easily implemented by users for specific purposes. Together with the seamless connection between the powerful computational and visual environment in R, it gives users more convenience and freedom to design figures for better understanding complex patterns behind multiple dimensional data. The package is described in Gu et al. 2014 <doi:10.1093/bioinformatics/btu393>.

URL https://github.com/jokergoo/circlize,

License MIT + file LICENSE

NeedsCompilation no

Repository CRAN

Date/Publication 2020-04-30 22:00:21 UTC
**R topics documented:**

<table>
<thead>
<tr>
<th>R topics documented:</th>
</tr>
</thead>
<tbody>
<tr>
<td>circlize-package</td>
</tr>
<tr>
<td>add_transparency</td>
</tr>
<tr>
<td>adjacencyList2Matrix</td>
</tr>
<tr>
<td>calc_gap</td>
</tr>
<tr>
<td>CELL_META</td>
</tr>
<tr>
<td>chordDiagram</td>
</tr>
<tr>
<td>chordDiagramFromDataFrame</td>
</tr>
<tr>
<td>chordDiagramFromMatrix</td>
</tr>
<tr>
<td>circlize</td>
</tr>
<tr>
<td>circos.arrow</td>
</tr>
<tr>
<td>circos.axis</td>
</tr>
<tr>
<td>circos.clear</td>
</tr>
<tr>
<td>circos.dendrogram</td>
</tr>
<tr>
<td>circos.genomicAxis</td>
</tr>
<tr>
<td>circos.genomicDensity</td>
</tr>
<tr>
<td>circos.genomicHeatmap</td>
</tr>
<tr>
<td>circos.genomicIdeogram</td>
</tr>
<tr>
<td>circos.genomicInitialize</td>
</tr>
<tr>
<td>circos.genomicLabels</td>
</tr>
<tr>
<td>circos.genomicLines</td>
</tr>
<tr>
<td>circos.genomicLink</td>
</tr>
<tr>
<td>circos.genomicPoints</td>
</tr>
<tr>
<td>circos.genomicPosTransformLines</td>
</tr>
<tr>
<td>circos.genomicRainfall</td>
</tr>
<tr>
<td>circos.genomicRect</td>
</tr>
<tr>
<td>circos.genomicText</td>
</tr>
<tr>
<td>circos.genomicTrack</td>
</tr>
<tr>
<td>circos.genomicTrackPlotRegion</td>
</tr>
<tr>
<td>circos.info</td>
</tr>
<tr>
<td>circos.initialize</td>
</tr>
<tr>
<td>circos.initializeWithIdeogram</td>
</tr>
<tr>
<td>circos.lines</td>
</tr>
<tr>
<td>circos.link</td>
</tr>
<tr>
<td>circos.nested</td>
</tr>
<tr>
<td>circos.par</td>
</tr>
<tr>
<td>circos.points</td>
</tr>
<tr>
<td>circospolygon</td>
</tr>
<tr>
<td>circos.raster</td>
</tr>
<tr>
<td>circos.rect</td>
</tr>
<tr>
<td>circos.segments</td>
</tr>
<tr>
<td>circos.text</td>
</tr>
<tr>
<td>circos.track</td>
</tr>
<tr>
<td>circos.trackHist</td>
</tr>
<tr>
<td>circos.trackLines</td>
</tr>
<tr>
<td>circos.trackPlotRegion</td>
</tr>
<tr>
<td>circos.trackPoints</td>
</tr>
</tbody>
</table>
R topics documented:

circos.trackText ................................................................. 75
circos.triangle ................................................................. 76
circos.update ................................................................. 77
circos.updatePlotRegion .................................................. 77
circos.xaxis ................................................................. 78
circos.yaxis ................................................................. 79
col2value ................................................................. 80
colorRamp2 ................................................................. 81
convert_height ............................................................. 82
convert_length .............................................................. 83
convert_x ................................................................. 84
convert_y ................................................................. 85
cytoband.col ............................................................... 86
degree ................................................................. 87
draw.sector ................................................................. 87
fontsize ................................................................. 89
generateRandomBed ....................................................... 89
genomicDensity .............................................................. 90
get.all.sector.index ...................................................... 91
get.all.track.index ........................................................ 91
get.cell.meta.data .......................................................... 92
get.current.chromosome .................................................. 93
get.current.sector.index ................................................. 94
get.current.track.index .................................................. 94
getI ................................................................. 95
highlight.chromosome .................................................... 95
highlight.sector ........................................................... 96
names.CELL_META .......................................................... 97
posTransform.default ..................................................... 98
posTransform.text ........................................................ 98
print.CELL_META ........................................................ 100
rainfallTransform .......................................................... 100
rand_color ................................................................. 101
read.chromInfo ............................................................ 102
read.cytoband ............................................................. 103
reverse.circlize ........................................................... 104
set.current.cell ............................................................ 105
show.index ................................................................. 106
smartAlign ................................................................. 106
uh ................................................................. 107
ux ................................................................. 107
uy ................................................................. 108
$.CELL_META ............................................................. 109

Index 110
Description
Circular visualization in R

Details
This package aims to implement circular layout in R.
Since most of the figures are composed of points, lines and polygons, we just need to implement low-level functions for drawing points, lines and polygons.
Current there are following low-level graphic functions:

- `circos.points`
- `circos.lines`
- `circos.rect`
- `circos.polygon`
- `circos.segments`
- `circos.text`
- `circos.axis, circos.xaxis, circos.yaxis`
- `circos.link`

For drawing points, lines and text through the whole track (among several sectors), the following functions are available:

- `circos.trackPoints`
- `circos.trackLines`
- `circos.trackText`

Functions to arrange circular layout:

- `circos.initialize`
- `circos.track`
- `circos.update`
- `circos.par`
- `circos.info`
- `circos.clear`

Theoretically, you are able to draw most kinds of circular plots by the above functions.
For specific use in genomics, we also implement functions which add graphics in genome scale.
Functions to initialize circos plot with genomic coordinates:
circos.initializeWithIdeogram

Functions to arrange genomic circular layout:

circos.genomicTrack

Functions to add basic graphics in genomic scale:

circos.genomicPoints
circos.genomicLines
circos.genomicText
circos.genomicRect
circos.genomicLink

Functions with specific purpose:

circos.genomicDensity
circos.genomicRainfall
circos.genomicIdeogram
circos.genomicHeatmap
circos.genomicLabels

Finally, function that draws Chord diagram:

chordDiagram

Please refer to the vignettes (http://jokergoo.github.io/circlize_book/book/) to find out how to draw basic and advanced circular plots by this package.

Examples

# There is no example
NULL
adjacencyList2Matrix

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>add_transparency</td>
<td>Add transparency to colors</td>
</tr>
<tr>
<td>adjacencyList2Matrix</td>
<td>Convert adjacency list to adjacency matrix</td>
</tr>
</tbody>
</table>

### add_transparency

**Description**

Add transparency to colors

**Usage**

```r
add_transparency(col, transparency = 0)
```

**Arguments**

- `col`: a vector of colors
- `transparency`: transparency, numeric value between 0 and 1

**Value**

A vector of colors

**Examples**

```r
add_transparency("red", 0.5)
add_transparency(1, 0.5)
add_transparency("#FF000080", 0.2)
```

### adjacencyList2Matrix

**Description**

Convert adjacency list to adjacency matrix

**Usage**

```r
adjacencyList2Matrix(lt, square = FALSE)
```

**Arguments**

- `lt`: a data frame which contains adjacency list.
- `square`: is the returned matrix a square matrix?

**Details**

Convert adjacency list to adjacency matrix.
Examples

# There is no example
NULL

Description

Calculate gap to make two Chord diagram with same scale

Usage

calc_gap(x1, x2, big.gap = 10, small.gap = 1)

Arguments

x1 The matrix or the data frame for the first Chord diagram.
x2 The matrix or the data frame for the second Chord diagram.
big.gap big.gap for the first Chord diagram.
small.gap small.gap for both Chord diagrams.

Details

There should be no overlap between the two sets of sectors.

Value

A numeric value which can be directly set to big.gap in the second Chord diagram.

Examples

set.seed(123)
mat1 = matrix(sample(20, 25, replace = TRUE), 5)
mat2 = mat1 / 2
gap = calc_gap(mat1, mat2, big.gap = 10, small.gap = 1)
chordDiagram(mat2, directional = 1, grid.col = rep(1:5, 2), transparency = 0.5,
             big.gap = gap, small.gap = 1)
**CELL_META**

*Easy way to get meta data in the current cell*

---

**Description**

Easy way to get meta data in the current cell

**Usage**

CELL_META

**Details**

The variable `CELL_META` can only be used to get meta data of the "current" cell. Basically you can simply replace e.g. `get.cell.meta.data("sector.index")` to `CELL_META$sector.index`.

**See Also**

`get.cell.meta.data`

**Examples**

```r
pdf(NULL)
circos.initialize("a", xlim = c(0, 1))
circos.track(ylim = c(0, 1), panel.fun = function(x, y) {
  print(CELL_META$sector.index)
  print(CELL_META$xlim)
})
print(names(CELL_META))
dev.off()
```

---

**chordDiagram**

*Plot Chord Diagram*

---

**Description**

Plot Chord Diagram

**Usage**

```r
chordDiagram(
  x,
  grid.col = NULL,
  grid.border = NA,
  transparency = 0.5,
  col = NULL,
)```
Arguments

x  
a matrix or a data frame. The function will pass all argument to `chordDiagramFromMatrix` or `chordDiagramFromDataFrame` depending on the type of `x`, also format of other arguments depends of the type of `x`. If it is in the form of a matrix, it should be an adjacency matrix. If it is in the form of a data frame, it should be an adjacency list.

grid.col  
pass to `chordDiagramFromMatrix` or `chordDiagramFromDataFrame`

grid.border  
pass to `chordDiagramFromMatrix` or `chordDiagramFromDataFrame`

transparency  
pass to `chordDiagramFromMatrix` or `chordDiagramFromDataFrame`

col  
pass to `chordDiagramFromMatrix` or `chordDiagramFromDataFrame`

row.col  
pass to `chordDiagramFromMatrix`

column.col  
pass to `chordDiagramFromMatrix`
order pass to `chordDiagramFromMatrix` or `chordDiagramFromDataFrame`
directional pass to `chordDiagramFromMatrix` or `chordDiagramFromDataFrame`
xmax maximum value on x-axes, the value should be a named vector.
symmetric pass to `chordDiagramFromMatrix`
keep.diagonal pass to `chordDiagramFromMatrix`
direction.type pass to `chordDiagramFromMatrix` or `chordDiagramFromDataFrame`
diffHeight pass to `chordDiagramFromMatrix` or `chordDiagramFromDataFrame`
reduce pass to `chordDiagramFromMatrix` or `chordDiagramFromDataFrame`
self.link pass to `chordDiagramFromMatrix` or `chordDiagramFromDataFrame`
preAllocateTracks pass to `chordDiagramFromMatrix` or `chordDiagramFromDataFrame`
annotationTrack pass to `chordDiagramFromMatrix` or `chordDiagramFromDataFrame`
annotationTrackHeight pass to `chordDiagramFromMatrix` or `chordDiagramFromDataFrame`
link.border pass to `chordDiagramFromMatrix` or `chordDiagramFromDataFrame`
link.lwd pass to `chordDiagramFromMatrix` or `chordDiagramFromDataFrame`
link.lty pass to `chordDiagramFromMatrix` or `chordDiagramFromDataFrame`
link.sort pass to `chordDiagramFromMatrix` or `chordDiagramFromDataFrame`
link.decreasing pass to `chordDiagramFromMatrix` or `chordDiagramFromDataFrame`
link.arr.length pass to `chordDiagramFromMatrix` or `chordDiagramFromDataFrame`
link.arr.width pass to `chordDiagramFromMatrix` or `chordDiagramFromDataFrame`
link.arr.type pass to `chordDiagramFromMatrix` or `chordDiagramFromDataFrame`
link.arr.lty pass to `chordDiagramFromMatrix` or `chordDiagramFromDataFrame`
link.arr.lwd pass to `chordDiagramFromMatrix` or `chordDiagramFromDataFrame`
link.arr.col pass to `chordDiagramFromMatrix` or `chordDiagramFromDataFrame`
link.largest.ontop pass to `chordDiagramFromMatrix` or `chordDiagramFromDataFrame`
link.visible pass to `chordDiagramFromMatrix` or `chordDiagramFromDataFrame`
link.rank order to add links to the circle, a large value means to add it later.
link.overlap pass to `chordDiagramFromMatrix` or `chordDiagramFromDataFrame`
scale scale each sector to same width
big.gap Gap between the two sets of sectors. If the input is a matrix, the two sets are row sectors and column sectors. If the input is a data frame, the two sets correspond to the first column and the second column. It only works when there is no intersection between the two sets.
small.gap Small gap between sectors.
... pass to `circos.link`
Details

Chord diagram is a way to visualize numeric tables (http://circos.ca/intro/tabular_visualization/), especially useful when the table represents information of directional relations. This function visualize tables in a circular way.

This function is flexible and contains some settings that may be a little difficult to understand. Please refer to vignette for better explanation.

Value

A data frame which contains positions of links, columns are:

- **rn** sector name corresponding to rows in the adjacency matrix or the first column in the adjacency list
- **cn** sector name corresponding to columns in the adjacency matrix or the second column in the adjacency list
- **value** value for the interaction or relation
- **o1** order of the link on the "from" sector
- **o2** order of the link on the "to" sector
- **x1** and position of the link on the "from" sector, the interval for the link on the "from" sector is $c(x1-\text{abs(value)},x1)$
- **x2** and position of the link on the "to" sector, the interval for the link on the "from" sector is $c(x2-\text{abs(value)},x2)$

See Also


Examples

```r
set.seed(999)
mat = matrix(sample(18, 18), 3, 6)
rownames(mat) = paste0("S", 1:3)
colnames(mat) = paste0("E", 1:6)

df = data.frame(from = rep(rownames(mat), times = ncol(mat)),
                to = rep(colnames(mat), each = nrow(mat)),
                value = as.vector(mat),
                stringsAsFactors = FALSE)

chordDiagram(mat)
chordDiagram(df)
circos.clear()
```
chordDiagramFromDataFrame

Plot Chord Diagram from a data frame

Description

Plot Chord Diagram from a data frame

Usage

```r
chordDiagramFromDataFrame(
  df,
  grid.col = NULL,
  grid.border = NA,
  transparency = 0.5,
  col = NULL,
  order = NULL,
  directional = 0,
  xmax = NULL,
  direction.type = "diffHeight",
  diffHeight = convert_height(2, "mm"),
  reduce = 1e-5,
  self.link = 2,
  preAllocateTracks = NULL,
  annotationTrack = c("name", "grid", "axis"),
  annotationTrackHeight = convert_height(c(3, 2), "mm"),
  link.border = NA,
  link.lwd = par("lwd"),
  link.lty = par("lty"),
  link.sort = FALSE,
  link.decreasing = TRUE,
  link.arr.length = ifelse(link.arr.type == "big.arrow", 0.02, 0.4),
  link.arr.width = link.arr.length/2,
  link.arr.type = "triangle",
  link.arr.lty = par("lty"),
  link.arr.lwd = par("lwd"),
  link.arr.col = par("col"),
  link.largest.on.top = FALSE,
  link.visible = TRUE,
  link.rank = seq_len(nrow(df)),
  link.overlap = FALSE,
  scale = FALSE,
  big.gap = 10,
  small.gap = 1,
  ...
)`
Arguments

**df**  
A data frame with at least two columns. The first two columns specify the connections and the third column (optional) contains numeric values which are mapped to the width of links as well as the colors if `col` is specified as a color mapping function. The sectors in the plot will be `union(df[[1]], df[[2]])`.

**grid.col**  
Grid colors which correspond to sectors. The length of the vector should be either 1 or the number of sectors. It’s preferred that `grid.col` is a named vector of which names correspond to sectors. If it is not a named vector, the order of `grid.col` corresponds to order of sectors.

**grid.border**  
border for grids. If it is `NULL`, the border color is same as grid color

**transparency**  
Transparency of link colors, 0 means no transparency and 1 means full transparency. If transparency is already set in `col` or `row.col` or `column.col`, this argument will be ignored. `NA` also ignores this argument.

**col**  
Colors for links. It can be a vector which corresponds to connections in `df`, or a function which generate colors according to values (the third column) in `df`, or a single value which means colors for all links are the same. You may use `colorRamp2` to generate a function which maps values to colors.

**order**  
Order of sectors. Default order is `union(df[[1]], df[[2]])`.

**directional**  
Whether links have directions. 1 means the direction is from the first column in `df` to the second column, -1 is the reverse, 0 is no direction, and 2 for two directional. The value can be a vector which has same length as number of rows in `df`.

**xmax**  
maximum value on x-axes, the value should be a named vector.

**direction.type**  
type for representing directions. Can be one or two values in “diffHeight” and “arrows”. If the value contains "diffHeight", different heights of the links are used to represent the directions for which starting root has long height to give people feeling that something is comming out. If the value contains "arrows", users can customize arrows with following arguments. The value can be a vector which has same length as number of rows in `df`. Note if you want to set both `diffHeight` and `arrows` for certain links, you need to embed these two options into one string such as "diffHeight+arrows".

**diffHeight**  
The difference of height between two 'roots' if `directional` is set to `TRUE`. If the value is set to a positive value, start root is shorter than end root and if it is set to a negative value, start root is longer than the end root. The value can be a vector which has same length as number of rows in `df`.

**reduce**  
if the ratio of the width of certain grid compared to the whole circle is less than this value, the grid is removed on the plot. Set it to value less than zero if you want to keep all tiny grid.

**self.link**  
if there is a self link in one sector, 1 means the link will be degenerated as a 'mountain' and the width corresponds to the value for this connection. 2 means the width of the starting root and the ending root all have the same width that corresponds to the value for the connection.

**preAllocateTracks**  
Pre-allocate empty tracks before drawing Chord diagram. It can be a single number indicating how many empty tracks needed to be created or a list containing settings for empty tracks. Please refer to vignette for details.
chordDiagramFromDataFrame

**annotationTrack**
Which annotation track should be plotted? By default, a track containing sector names and a track containing grid will be created.

**annotationTrackHeight**
Track height corresponding to values in `annotationTrack`.

**link.border**
border for links, single scalar or a vector which has the same length as `nrows` of `df` or a data frame

**link.lwd**
width for link borders, single scalar or a vector which has the same length as `nrows` of `df` or a data frame

**link.lty**
style for link borders, single scalar or a vector which has the same length as `nrows` of `df` or a data frame

**link.sort**
whether sort links on every sector based on the width of the links on it. If it is set to "overall", all links are sorted regardless whether they are from the first column or the second column.

**link.decreasing**
for `link.sort`

**link.arr.length**
pass to `circos.link`. The format of this argument is same as `link.lwd`.

**link.arr.width**
pass to `Arrowhead`. The format of this argument is same as `link.lwd`.

**link.arr.type**
pass to `circos.link`, same settings as `link.lwd`. Default value is `triangle`.

**link.arr.col**
color or the single line link which is put in the center of the belt. The format of this argument is same as `link.lwd`.

**link.arr.lwd**
line width of the single line link which is put in the center of the belt. The format of this argument is same as `link.lwd`.

**link.arr.lty**
line type of the single line link which is put in the center of the belt. The format of this argument is same as `link.lwd`.

**link.largest.ontop**
controls the order of adding links, whether based on the absolute value?

**link.visible**
whether plot the link. The value is logical, if it is set to `FALSE`, the corresponding link will not plotted, but the space is still occupied. The format of this argument is same as `link.lwd`.

**link.rank**
order to add links to the circle, a large value means to add it later.

**link.overlap**
if it is a directional Chord Diagram, whether the links that come or end in a same sector overlap?

**scale**
scale each sector to same width

**big.gap**
Gaps between the sectors in the first column of `df` and sectors in the second column in `df`.

**small.gap**
Small gap between sectors.

**...**
pass to `circos.link`

**Details**
The data frame can have a column named "rank" which is used to control the order of adding links to the diagram.
chordDiagramFromMatrix

Value

A data frame which contains positions of links, see explanation in chordDiagram.

See Also


Examples

# There is no example
NULL
link.decreasing = TRUE,
link.arr.length = ifelse(link.arr.type == "big.arrow", 0.02, 0.4),
link.arr.width = link.arr.length/2,
link.arr.type = "triangle",
link.arr.lty = par("lty"),
link.arr.lwd = par("lwd"),
link.arr.col = par("col"),
link.largest.ontop = FALSE,
link.visible = TRUE,
link.rank = NULL,
link.overlap = FALSE,
scale = FALSE,
big.gap = 10,
small.gap = 1,
...)

Arguments

mat A table which represents as a numeric matrix.
grid.col Grid colors which correspond to matrix rows/columns (or sectors). The length of the vector should be either 1 or length(union(rownames(mat),colnames(mat))). It's preferred that grid.col is a named vector of which names correspond to sectors. If it is not a named vector, the order of grid.col corresponds to order of sectors.
grid.border border for grids. If it is NULL, the border color is same as grid color.
transparency Transparency of link colors, 0 means no transparency and 1 means full transparency. If transparency is already set in col or row.col or column.col, this argument will be ignored. NA also ignores this argument.
col Colors for links. It can be a matrix which corresponds to mat, or a function which generate colors according to values in mat, or a single value which means colors for all links are the same, or a three-column data frame in which the first two columns correspond to row names and columns and the third column is colors. You may use colorRamp2 to generate a function which maps values to colors.
row.col Colors for links. Links from the same row in mat will have the same color. Length should be same as number of rows in mat. This argument only works when col is set to NULL.
column.col Colors for links. Links from the same column in mat will have the same color. Length should be same as number of columns in mat. This argument only works when col and row.col is set to NULL.
order Order of sectors. Default order is union(df[[1]],df[[2]]).
directional Whether links have directions. 1 means the direction is from the first column in df to the second column, -1 is the reverse, 0 is no direction, and 2 for two directional. Same setting as link.border.
xmax maximum value on x-axes, the value should be a named vector.
direction.type  type for representing directions. Can be one or two values in "diffHeight" and "arrows". If the value contains "diffHeight", different heights of the links are used to represent the directions for which starting root has long height to give people feeling that something is coming out. If the value contains "arrows", users can customize arrows with following arguments. Same setting as link.border. Note if you want to set both diffHeight and arrows for certain links, you need to embed these two options into one string such as "diffHeight+arrows".

diffHeight  The difference of height between two 'roots' if directional is set to TRUE. If the value is set to a positive value, start root is shorter than end root and if it is set to a negative value, start root is longer than the end root.

reduce  if the ratio of the width of certain grid compared to the whole circle is less than this value, the grid is removed on the plot. Set it to value less than zero if you want to keep all tiny grid.

self.link  if there is a self link in one sector, 1 means the link will be degenerated as a 'mountain' and the width corresponds to the value for this connection. 2 means the width of the starting root and the ending root all have the width that corresponds to the value for the connection.

symmetric  Whether the matrix is symmetric. If the value is set to TRUE, only lower triangular matrix without the diagonal will be used.

keep.diagonal  If the matrix is specified as symmetric, whether keep diagonal for visualization.

preAllocateTracks  Pre-allocate empty tracks before drawing Chord diagram. It can be a single number indicating how many empty tracks needed to be created or a list containing settings for empty tracks. Please refer to vignette for details.

annotationTrack  Which annotation track should be plotted? By default, a track containing sector names and a track containing grid will be created.

annotationTrackHeight  Track height corresponding to values in annotationTrack.

link.border  border for links, single scalar or a matrix with names or a data frame with three columns

link.lwd  width for link borders, single scalar or a matrix with names or a data frame with three columns

link.lty  style for link borders, single scalar or a matrix with names or a data frame with three columns

link.sort  whether sort links on every sector based on the width of the links on it. If it is set to "overall", all links are sorted regardless whether they are from rows or columns.

link.decreasing  for link.sort

link.arr.length  pass to circos.link. The format of this argument is same as link.lwd.

link.arr.width  pass to Arrowhead. The format of this argument is same as link.lwd.

link.arr.type  pass to circos.link, same format as link.lwd. Default value is triangle.
link.arr.col  color or the single line link which is put in the center of the belt. The format of this argument is same as link.lwd.

link.arr.lwd  line width of the single line link which is put in the center of the belt. The format of this argument is same as link.lwd.

link.arr.lty  line type of the single line link which is put in the center of the belt. The format of this argument is same as link.lwd.

link.largest.ontop  controls the order of adding links, whether based on the absolute value?

link.visible  whether plot the link. The value is logical, if it is set to FALSE, the corresponding link will not plotted, but the space is still occupied. The format of this argument is same as link.lwd

link.rank  order to add links to the circle, a large value means to add it later.

link.overlap  if it is a directional Chord Diagram, whether the links that come or end in a same sector overlap?

scale  scale each sector to same width

big.gap  Gap between row sectors and column sectors.

small.gap  Small gap between sectors.

...  pass to circos.link

Details

Internally, the matrix is transformed to a data frame and sent to chordDiagramFromDataFrame.

Value

A data frame which contains positions of links, see explanation in chordDiagram.

See Also


Examples

# There is no example
NULL
circlize  

Convert to polar coordinate system

Description

Convert to polar coordinate system

Usage

```
circlize(
  x, y,
  sector.index = get.current.sector.index(),
  track.index = get.current.track.index())
```

Arguments

- **x**: Data points on x-axis. The value can also be a two-column matrix/data frame if you put x and y data points into one variable.
- **y**: Data points on y-axis.
- **sector.index**: Index for the sector to convert the coordinates
- **track.index**: Index for the track to convert the coordinates

Details

This is the core function in the package. It transforms data points from data coordinate system (in a specific cell) to the polar coordinate system.

Value

A matrix with two columns (theta and rou). rou is measured in degree.

Examples

```
pdf(NULL)
factors = c("a", "b")
circos.initialize(factors, xlim = c(0, 1))
circos.track(ylim = c(0, 1))
# x = 0.5, y = 0.5 in sector a and track 1
circlize(0.5, 0.5, sector.index = "a", track.index = 1)
circos.clear()
dev.off()
```


The `circos.arrow` function is used to draw an arrow which is parallel to the circle in a `circos` plot.

### Description

**Draw arrow which is parallel to the circle**

### Usage

```r
circos.arrow(
x1,
x2,
y = get.cell.meta.data("ycenter", sector.index, track.index),
width = get.cell.meta.data("yrange", sector.index, track.index)/2,
sector.index = get.current.sector.index(),
track.index = get.current.track.index(),
arrow.head.length = convert_x(5, "mm", sector.index, track.index),
arrow.head.width = width*2,
arrow.position = c("end", "start"),
tail = c("normal", "point"),
border = "black",
col = "white",
lty = par("lty"),
...)
```

### Arguments

- **x1**: start position of the arrow on the x-axis.
- **x2**: end position of the arrow on the x-axis.
- **y**: position of the arrow on the y-axis. Note this is the center of the arrow on y-axis.
- **width**: width of the arrow body.
- **sector.index**: index of the sector.
- **track.index**: index of the track.
- **arrow.head.length**: length of the arrow head. Note the value should be smaller than the length of the arrow itself (which is `x2 - x1`).
- **arrow.head.width**: width of the arrow head.
- **arrow.position**: where is the arrow head on the arrow.
- **tail**: the shape of the arrow tail (the opposite side of arrow head).
- **border**: border color of the arrow.
- **col**: filled color of the arrow.
- **lty**: line style of the arrow.
- **...**: pass to `polygon`. 
Details

Note all position values are measured in the data coordinate (the coordinate in each cell).

If you see points overflow warnings, you can set `circos.par(points.overflow.warning = FALSE)` to turn it off.

Author(s)

Zuguang Gu <z.gu@dkfz.de>

See Also


Examples

```r
circos.initialize(letters[1:4], xlim = c(0, 1))
circos.track(ylim = c(0, 1), panel.fun = function(x, y) {
  circos.arrow(0, 1, y = 0.5, width = 0.4, arrow.head.length = ux(1, "cm"),
  col = "red", tail = ifelse(CELL_META$sector.index %in% c("a", "c"),
  "point", "normal"))
}, bg.border = NA, track.height = 0.4)
circos.clear()

########## cell cycle ###########
cell_cycle = data.frame(phase = factor(c("G1", "S", "G2", "M"),
  levels = c("G1", "S", "G2", "M")),
  hour = c(11, 8, 4, 1))
color = c("#66C2A5", "#FC8D62", "#8DA0CB", "#E78AC3")
circos.par(start.degree = 90)
circos.initialize(cell_cycle$phase, xlim = cbind(rep(0, 4), cell_cycle$hour))
circos.track(ylim = c(0, 1), panel.fun = function(x, y) {
  circos.arrow(CELL_META$xlim[1], CELL_META$xlim[2],
  arrow.head.width = CELL_META$yrange*0.8, arrow.head.length = ux(1, "cm"),
  col = color(CELL_META$sector.numeric.index))
circos.text(CELL_META$xcenter, CELL_META$ycenter, CELL_META$sector.index,
  facing = "downward")
}, bg.border = NA, track.height = 0.3)
circos.clear()
```

---

**circos.axis**

__Draw x-axis__

**Description**

Draw x-axis
Usage

circos.axis(
    h = "top",
    major.at = NULL,
    labels = TRUE,
    major.tick = TRUE,
    sector.index = get.cell.meta.data("sector.index"),
    track.index = get.cell.meta.data("track.index"),
    labels.font = par("font"),
    labels.cex = par("cex"),
    labels.facing = "inside",
    labels.direction = NULL,
    labels.niceFacing = TRUE,
    direction = c("outside", "inside"),
    minor.ticks = 4,
    major.tick.percentage = 0.1,
    labels.away.percentage = major.tick.percentage/2,
    major.tick.length = convert_y(1, "mm", sector.index, track.index),
    lwd = par("lwd"),
    col = par("col"),
    labels.col = par("col"),
    labels.pos.adjust = TRUE)

Arguments

h Position of the x-axis, can be "top", "bottom" or a numeric value
major.at If it is numeric vector, it identifies the positions of the major ticks. It can exceed
xlim value and the exceeding part would be trimmed automatically. If it is NULL,
about every 10 degrees there is a major tick.
labels labels of the major ticks. Also, the exceeding part would be trimmed automat-
ically. The value can also be logical (either an atomic value or a vector) which
represents which labels to show.
major.tick Whether to draw major tick. If it is set to FALSE, there would be no minor ticks.
sector.index Index for the sector
track.index Index for the track
labels.font font style for the axis labels
labels.cex font size for the axis labels
labels.direction deprecated, use facing instead.
labels.facing facing of labels on axis, passing to circos.text
labels.niceFacing Should facing of axis labels be human-easy
direction whether the axis ticks point to the outside or inside of the circle.
minor.ticks Number of minor ticks between two close major ticks.
major.tick.percentage
not used. Length of the major ticks. It is the percentage to the height of the cell.

labels.away.percentage
not used. The distance for the axis labels to the major ticks. It is the percentage to the height of the cell.

major.tick.length
length of the major ticks, measured in "current" data coordinate. convert_y can be used to convert an absolute unit to the data coordinate.

lwd
line width for ticks
col
color for the axes
labels.col
color for the labels
labels.pos.adjust
whether to adjust the positions of the first label and the last label. The value can be a vector of length two which correspond to the first label and the last label.

Details
It can only draw axes on x-direction.

See Also
circos.yaxis draws axes on y-direction.


Examples

factors = letters[1:8]
circos.par(points.overflow.warning = FALSE)
circos.initialize(factors = factors, xlim = c(0, 10))
circos.trackPlotRegion(factors = factors, ylim = c(0, 10), track.height = 0.1,
    bg.border = NA, panel.fun = function(x, y) {
        circos.text(5, 10, get.cell.meta.data("sector.index"))
    })
}
circos.trackPlotRegion(factors = factors, ylim = c(0, 10))
circos.axis(sector.index = "a")
circos.axis(sector.index = "b", direction = "inside", labels.facing = "outside")
circos.axis(sector.index = "c", h = "bottom")
circos.axis(sector.index = "d", h = "bottom", direction = "inside",
    labels.facing = "reverse.clockwise")
circos.axis(sector.index = "e", h = 5, major.at = c(1, 3, 5, 7, 9))
circos.axis(sector.index = "f", h = 5, major.at = c(1, 3, 5, 7, 9),
    labels = c("a", "c", "e", "g", "f"), minor.ticks = 0)
circos.axis(sector.index = "g", h = 5, major.at = c(1, 3, 5, 7, 9),
    labels = c("a1", "c1", "e1", "g1", "f1"), major.tick = FALSE,
    labels.facing = "reverse.clockwise")
circos.axis(sector.index = "h", h = 2, major.at = c(1, 3, 5, 7, 9),
    labels = c("a1", "c1", "e1", "g1", "f1"), major.tick.percentage = 0.3,
    labels.away.percentage = 0.2, minor.ticks = 2, labels.facing = "clockwise")
circos.clear()
if(FALSE) {

 ######################## real-time clock ########################
 factors = letters[1]

circos.par("gap.degree" = 0, "cell.padding" = c(0, 0, 0, 0), "start.degree" = 90)
circos.initialize(factors = factors, xlim = c(0, 12))
circos.trackPlotRegion(factors = factors, ylim = c(0, 1), bg.border = NA)
circos.axis(sector.index = "a", major.at = 0:12, labels = "",
            direction = "inside", major.tick.percentage = 0.3)
circos.text(1:12, rep(0.5, 12), 1:12, facing = "downward")

while(1) {
  current.time = as.POSIXlt(Sys.time())
  sec = ceiling(current.time$sec)
  min = current.time$min
  hour = current.time$hour

  # erase the clock hands
  draw.sector(rou1 = 0.8, border = "white", col = "white")

  sec.degree = 90 - sec/60 * 360
  arrows(0, 0, cos(sec.degree/180*pi)*0.8, sin(sec.degree/180*pi)*0.8)

  min.degree = 90 - min/60 * 360
  arrows(0, 0, cos(min.degree/180*pi)*0.7, sin(min.degree/180*pi)*0.7, lwd = 2)

  hour.degree = 90 - hour/12 * 360 - min/60 * 360/12
  arrows(0, 0, cos(hour.degree/180*pi)*0.4, sin(hour.degree/180*pi)*0.4, lwd = 2)

  Sys.sleep(1)
}
circos.clear()
}

---
circos.clear
Reset the circular layout parameters

Description
Reset the circular layout parameters

Usage
circos.clear()
Details

Because there are several parameters for the circular plot which can only be set before `circos.initialize`. So before you draw the next circular plot, you need to reset all these parameters.

If you meet some errors when re-drawing the circular plot, try running this function and it will solve most of the problems.

Examples

```r
# There is no example
NULL
```

Description

Add circular dendrograms

Usage

```r
circos.dendrogram(
  dend,
  facing = c("outside", "inside"),
  max_height = NULL,
  use_x_attr = FALSE)
```

Arguments

- `dend` A `dendrogram` object.
- `facing` Is the dendrograms facing inside to the circle or outside.
- `max_height` Maximum height of the dendrogram. This is important if more than one dendrograms are drawn in one track and making them comparable.
- `use_x_attr` Whether use the x attribute to determine node positions in the dendrogram, used internally.

Details

Assuming there are n nodes in the dendrogram, the positions for leaves on x-axis is 0.5, 1.5, ..., n - 0.5. So you must be careful with `xlim` when you initialize the circular layout.

You can use the dendextend package to render the dendrograms.

See Also

Examples

```r
load(system.file(package = "circlize", "extdata", "bird.orders.RData"))

labels = hc$labels # name of birds
c = cutree(hc, 6) # cut tree into 6 pieces
n = length(labels) # number of bird species
dend = as.dendrogram(hc)

circos.par(cell.padding = c(0, 0, 0, 0))
circos.initialize(factors = "a", xlim = c(0, n)) # only one sector
max_height = attr(dend, "height") # maximum height of the trees
circos.trackPlotRegion(ylim = c(0, 1), bg.border = NA, track.height = 0.3,
  panel.fun = function(x, y) {
    for(i in seq_len(n)) {
      circos.text(i-0.5, 0, labels[i], adj = c(0, 0.5),
        facing = "clockwise", niceFacing = TRUE,
        col = ct[labels[i]], cex = 0.7)
    }
  })
}

suppressPackageStartupMessages(require(dendextend))
dend = color_branches(dend, k = 6, col = 1:6)

circos.trackPlotRegion(ylim = c(0, max_height), bg.border = NA,
  track.height = 0.4, panel.fun = function(x, y) {
    circos.dendrogram(dend, max_height = max_height)
  })
circos.clear()
```

circos.genomicAxis Add genomic axes

Description

Add genomic axes

Usage

```r
circos.genomicAxis(  
  h = "top",  
  major.at = NULL,  
  labels = NULL,  
  major.by = NULL,  
  tickLabelsStartFromZero = TRUE,  
  labels.cex = 0.4*par("cex"),  
  sector.index = get.cell.meta.data("sector.index"),  
  track.index = get.cell.meta.data("track.index"),  
  ...)
```
Arguments

- `h` (Position of the axes. "top" or "bottom").
- `major.at` (Major breaks. If `major.at` is set, `major.by` is ignored).
- `labels` (labels corresponding to `major.at`. If `labels` is set, `major.at` must be set).
- `major.by` (Increment of major ticks. It is calculated automatically if the value is not set.
  (about every 10 degrees there is a major tick).
- `tickLabelsStartFromZero` (Whether axis tick labels start from 0? This will only affect the axis labels while not affect x-values in cells).
- `labels.cex` (the font size for the axis tick labels).
- `sector.index` (Index for the sector).
- `track.index` (Index for the track).
- `...` (Other arguments pass to `circos.axis`).

Details

It assigns proper tick labels under genomic coordinate.

See Also


Examples

```r
circos.initializeWithIdeogram(plotType = NULL)
circos.track(ylim = c(0, 1), panel.fun = function(x, y) circos.genomicAxis())
circos.clear()
```

circos.genomicDensity  
Calculate and add genomic density track

Description

Calculate and add genomic density track

Usage

```r
circos.genomicDensity(
  data,
  ylim.force = FALSE,
  window.size = NULL,
  overlap = TRUE,
  count_by = c("percent", "number"),
  col = ifelse(area, "grey", "black"),
```

lwd = par("lwd"),
lty = par("lty"),
type = "l",
area = TRUE,
area.baseline = NULL,
baseline = 0,
border = NA,
...

Arguments

- **data**: A bed-file-like data frame or a list of data frames
- **ylim.force**: Whether to force upper bound of ylim to be 1.
- **window.size**: Pass to `genomicDensity`
- **overlap**: Pass to `genomicDensity`
- **count_by**: Pass to `genomicDensity`
- **col**: Colors. It should be length of one. If data is a list of data frames, the length of col can also be the length of the list.
- **lwd**: Width of lines
- **lty**: Style of lines
- **type**: Type of lines, see `circos.lines`
- **area**: See `circos.lines`
- **area.baseline**: Deprecated, use baseline instead.
- **baseline**: See `circos.lines`
- **border**: See `circos.lines`
- ... Pass to `circos.trackPlotRegion`

Details

This function is a high-level graphical function, and it will create a new track.

See Also


Examples

```r
load(system.file(package = "circlize", "extdata", "DMR.RData"))

# rainfall
## Not run:
circos.initializeWithIdeogram(plotType = c("axis", "labels"))

bed_list = list(DMR_hyper, DMR_hypo)
circos.genomicRainfall(bed_list, pch = 16, cex = 0.4, col = c("#FF000080", "#0000FF80"))
```


circos.genomicDensity(bed_list[[1]], col = c("#FF000080"), track.height = 0.1)
circos.genomicDensity(bed_list[[2]], col = c("#0000FF80"), track.height = 0.1)

circos.clear()

## End(Not run)

---

circos.genomicHeatmap  Add heatmaps for selected regions

Description
Add heatmaps for selected regions

Usage

circos.genomicHeatmap(
    bed,
    col,
    na_col = "grey",
    numeric.column = NULL,
    border = NA,
    border_lwd = par("lwd"),
    border_lty = par("lty"),
    connection_height = convert_height(5, "mm"),
    line_col = par("col"),
    line_lwd = par("lwd"),
    line_lty = par("lty"),
    heatmap_height = 0.15,
    side = c("inside", "outside"),
    track.margin = circos.par("track.margin"))

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>bed</td>
<td>a data frame in bed format, the matrix is stored from the fourth column.</td>
</tr>
<tr>
<td>col</td>
<td>colors for the heatmaps. The value can be a matrix or a color mapping function generated by <code>colorRamp2</code>.</td>
</tr>
<tr>
<td>na_col</td>
<td>color for NA values.</td>
</tr>
<tr>
<td>numeric.column</td>
<td>column index for the numeric columns. The values can be integer index or character index</td>
</tr>
<tr>
<td>border</td>
<td>border of the heatmap grids.</td>
</tr>
<tr>
<td>border_lwd</td>
<td>line width for borders of heatmap grids</td>
</tr>
<tr>
<td>border_lty</td>
<td>line style for borders of heatmap grids</td>
</tr>
<tr>
<td>connection_height</td>
<td>height of the connection lines. If it is set to NULL, no connection will be drawn.</td>
</tr>
</tbody>
</table>
line_col col of the connection line. The value can be a vector.
line_lwd line width of the connection lines.
line_lty line style of the connection lines.
heatmap_height height of the heatmap track
side side of the heatmaps. Is the heatmap facing inside or outside?
track.margin bottom and top margins

Details

The function visualizes heatmaps which correspond to a subset of regions in the genome. The correspondence between heatmaps and regions are identified by connection lines.

The function actually creates two tracks, one track for the connection lines and one track for the heatmaps. The heatmaps always fill the whole track.

See Also


Examples

## Not run:
circos.initializeWithIdeogram(plotType = c("labels", "axis"))
bed = generateRandomBed(nr = 100, nc = 4)
col_fun = colorRamp2(c(-1, 0, 1), c("green", "black", "red"))
circos.genomicHeatmap(bed, col_fun, side = "inside", border = "white")
circos.genomicHeatmap(bed, col_fun, side = "outside",
                      line_col = as.numeric(factor(bed[[1]])))

## End(Not run)

circos.genomicIdeogram

Add an ideogram track

Description

Add an ideogram track

Usage

circos.genomicIdeogram(
    cytoband = system.file(package = "circlize", "extdata", "cytoBand.txt"),
    species = NULL,
    track.height = convert_height(2, "mm"),
    track.margin = circos.par("track.margin"))
Initialize circular plot with any genomic data

circos.genomicInitialize(data, sector.names = NULL, major.by = NULL, plotType = c("axis", "labels"), tickLabelsStartFromZero = TRUE, axis.labels.cex = 0.4*par("cex"), labels.cex = 0.8*par("cex"), track.height = NULL, ...)

Description

Initialize circular plot with any genomic data
Arguments

data | A data frame containing genomic data.
sector.names | Labels for each sectors which will be drawn along each sector. It will not modify values of sector index.
major.by | Increment of major ticks. It is calculated automatically if the value is not set (about every 10 degrees there is a major tick).
plotType | If it is not NULL, there will create a new track containing axis and names for sectors. This argument controls which part should be drawn, axis for genomic axis and labels for chromosome names.
tickLabelsStartFromZero | Whether axis tick labels start from 0? This will only affect the axis labels while not affect x-values in cells.
axis.labels.cex | the font size for the axis tick labels.
labels.cex | the font size for the labels.
track.height | If PlotType is not NULL, height of the annotation track.
... | Pass to circos.initialize

Details

The function will initialize circular plot from genomic data. If plotType is set with value in axis or labels, there will create a new track.
The order of sectors related to data structure of data. If the first column in data is a factor, the order of sectors is levels(data[[1]]); If the first column is just a simple vector, the order of sectors is unique(data[[1]]).
For more details on initializing genomic plot, please refer to the vignettes.

See Also


Examples

df = read.cytoband()$df
circos.genomicInitialize(df)

df = data.frame(name = c("TP53", "TP63", "TP73"),
               start = c(7565097, 189349205, 3569084),
               end = c(7590856, 189615068, 3652765),
               stringsAsFactors = FALSE)
circos.genomicInitialize(df)
circos.clear()

circos.genomicInitialize(df, major.by = 10000)
circos.clear()

circos.genomicInitialize(df, plotType = "labels")
circos.clear()
circos.genomicInitialize(df, sector.names = c("tp53", "tp63", "tp73"))
circos.clear()
circos.genomicInitialize(df, sector.names = c("tp53x", "tp63x", "tp73"))
circos.clear()
df[[1]] = factor(df[[1]], levels = c("TP73", "TP63", "TP53"))
circos.genomicInitialize(df)
circos.clear()

circos.genomicLabels

\textit{Add labels to specified genomic regions}

\section*{Description}
Add labels to specified genomic regions

\section*{Usage}

\begin{verbatim}
circos.genomicLabels(    bed,    labels = NULL,    labels.column = NULL,    facing = "clockwise",    niceFacing = TRUE,    col = par("col"),    cex = 0.8,    font = par("font"),    padding = 0.4,    connection_height = convert_height(5, "mm"),    line_col = par("col"),    line_lwd = par("lwd"),    line_lty = par("lty"),    labels_height = min(c(convert_height(1.5, "cm"), max(strwidth(labels, cex = cex, font = font)))))
                       ,    side = c("inside", "outside"),    track.margin = circos.par("track.margin"))
\end{verbatim}

\section*{Arguments}
\begin{itemize}
\item \textbf{bed} a data frame in bed format
\item \textbf{labels} a vector of labels corresponding to rows in \textit{bed}
\item \textbf{labels.column} if the label column is already in \textit{bed}, the index for this column in \textit{bed}
\item \textbf{facing} facing of the labels. The value can only be ‘clockwise’ or ‘reverse.clockwise’.
\item \textbf{niceFacing} whether automatically adjust the facing of the labels.
\end{itemize}
The function adds labels for the specified regions. The positions of labels are arranged so that they are not overlapping to each other.

See Also


Examples

circos.initializeWithIdeogram(plotType = c("labels", "axis"))
bed = generateRandomBed(nr = 100, fun = function(k) sample(letters, k, replace = TRUE))
bed[1, 4] = "aaaaaaaa"
circos.genomicLabels(bed, labels.column = 4, side = "inside",
  col = as.numeric(factor(bed[[1]])))
circos.genomicLabels(bed, labels.column = 4, side = "outside",
  line_col = as.numeric(factor(bed[[1]])))


circos.genomicLines Add lines to a plotting region, specifically for genomic graphics

Description

Add lines to a plotting region, specifically for genomic graphics
Usage

circos.genomicLines(
  region,
  value,
  numeric.column = NULL,
  sector.index = get.cell.meta.data("sector.index"),
  track.index = get.cell.meta.data("track.index"),
  posTransform = NULL,
  col = ifelse(area, "grey", "black"),
  lwd = par("lwd"),
  lty = par("lty"),
  type = "l",
  area = FALSE,
  area.baseline = NULL,
  border = "black",
  baseline = "bottom",
  pt.col = par("col"),
  cex = par("cex"),
  pch = par("pch"),
  ...
)

Arguments

region A data frame contains 2 column which correspond to start position and end position
value A data frame contains values and other information
numeric.column Which column in value data frame should be taken as y-value. If it is not defined, the whole numeric columns in value will be taken.
sector.index Pass to circos.lines
track.index Pass to circos.lines
posTransform Self-defined function to transform genomic positions, see posTransform.default for explanation
col col of lines/areas. If there are more than one numeric column, the length of col can be either one or number of numeric columns. If there is only one numeric column and type is either segment or h, the length of col can be either one or number of rows of region. pass to circos.lines
lwd Settings are similar as col. Pass to circos.lines
lty Settings are similar as col. Pass to circos.lines
type There is an additional option segment which plot segment lines from start position to end position. Settings are similar as col. Pass to circos.lines.area Settings are similar as col. Pass to circos.lines.area.baseline Deprecated, use baseline instead.
baseline Settings are similar as col. Pass to circos.lines.border Settings are similar as col. Pass to circos.lines
The function is a low-level graphical function and usually is put in panel.fun when using `circos.genomicTrackPlotRegion`.

Examples

```r
### test bed
circos.par("track.height" = 0.1)
circos.initializeWithIdeogram(plotType = NULL)

bed = generateRandomBed(nr = 100)
circos.genomicTrackPlotRegion(bed, panel.fun = function(region, value, ...) {
circos.genomicLines(region, value, type = "l", ...) 
})

bed1 = generateRandomBed(nr = 100)
bed2 = generateRandomBed(nr = 100)
bed_list = list(bed1, bed2)
circos.genomicTrackPlotRegion(bed_list, panel.fun = function(region, value, ...) {
i = getI(...)
circos.genomicLines(region, value, col = i, ...)
})
circos.genomicTrackPlotRegion(bed_list, stack = TRUE, 
panel.fun = function(region, value, ...) {
i = getI(...)
circos.genomicLines(region, value, col = i, ...)
})

bed = generateRandomBed(nr = 100, nc = 4)
circos.genomicTrackPlotRegion(bed, panel.fun = function(region, value, ...) {
circos.genomicLines(region, value, col = 1:4, ...)
})
circos.genomicTrackPlotRegion(bed, stack = TRUE, panel.fun = function(region, value, ...) {
i = getI(...)
circos.genomicLines(region, value, col = i, ...)
})

bed = generateRandomBed(nr = 100)
circos.genomicTrackPlotRegion(bed, panel.fun = function(region, value, ...) {
circos.genomicLines(region, value, type = "segment", lwd = 2, ...)
})
circos.clear()
```
Add links from two sets of genomic positions

Usage
circos.genomicLink(
  region1,
  region2,
  rou = get_most_inside_radius(),
  rou1 = rou,
  rou2 = rou,
  col = "black",
  lwd = par("lwd"),
  lty = par("lty"),
  border = col,
  ...
)

Arguments
region1 A genomic data frame
region2 A genomic data frame
rou Pass to circos.link
rou1 Pass to circos.link
rou2 Pass to circos.link
col Pass to circos.link, length can be either one or nrow of region1
lwd Pass to circos.link, length can be either one or nrow of region1
lty Pass to circos.link, length can be either one or nrow of region1
border Pass to circos.link, length can be either one or nrow of region1
...

Details
Of course, number of rows should be same in region1 and region2.
If you want to have more controls on links, please use circos.link directly.

See Also
### Examples

```r
## Not run:
set.seed(123)

bed1 = generateRandomBed(nr = 100)
bed1 = bed1[sample(nrow(bed1), 20), ]
bed2 = generateRandomBed(nr = 100)
bed2 = bed2[sample(nrow(bed2), 20), ]
circos.par("track.height" = 0.1, cell.padding = c(0, 0, 0, 0))
circos.initializeWithIdeogram()

circos.genomicLink(bed1, bed2, col = sample(1:5, 20, replace = TRUE), border = NA)
circos.clear()

## End(Not run)
```

---

circos.genomicPoints  
*Add points to a plotting region, specifically for genomic graphics*

#### Description

Add points to a plotting region, specifically for genomic graphics

#### Usage

```r
circos.genomicPoints(
  region,
  value,
  numeric.column = NULL,
  sector.index = get.cell.meta.data("sector.index"),
  track.index = get.cell.meta.data("track.index"),
  posTransform = NULL,
  pch = par("pch"),
  col = par("col"),
  cex = par("cex"),
  bg = par("bg"),
  ...
)
```

#### Arguments

- **region**: A data frame contains 2 columns which correspond to start positions and end positions
- **value**: A data frame contains values and other information
- **numeric.column**: Which column in value data frame should be taken as y-value. If it is not defined, the whole numeric columns in value will be taken.
- **sector.index**: Pass to `circos.points`
- **track.index**: Pass to `circos.points`
circos.genomicPoints

posTransform  Self-defined function to transform genomic positions, see posTransform.default for explanation

col  color of points. If there is only one numeric column, the length of col can be either one or number of rows of region. If there are more than one numeric column, the length of col can be either one or number of numeric columns. Pass to circos.points

pch  Type of points. Settings are similar as col. Pass to circos.points

cex  Size of points. Settings are similar as col. Pass to circos.points

bg  background colors for points.

...  Mysterious parameters

Details

The function is a low-level graphical function and usually is put in panel.fun when using circos.genomicTrackPlotRegion.

Examples

circcos.par("track.height" = 0.1)
circcos.initializeWithIdeogram(plotType = NULL)

bed = generateRandomBed(nr = 100)
circcos.genomicTrackPlotRegion(bed, panel.fun = function(region, value, ...) {
  circcos.genomicPoints(region, value, pch = 16, cex = 0.5, ...)
})

circcos.genomicTrackPlotRegion(bed, stack = TRUE, panel.fun = function(region, value, ...) {
  circcos.genomicPoints(region, value, pch = 16, cex = 0.5, ...)
  i = getI(...)  
  cell.xlim = get.cell.meta.data("cell.xlim")
  circcos.lines(cell.xlim, c(i, i), lty = 2, col = "#00000040")
})

bed1 = generateRandomBed(nr = 100)
bed2 = generateRandomBed(nr = 100)
bed_list = list(bed1, bed2)

# data frame list
circcos.genomicTrackPlotRegion(bed_list, panel.fun = function(region, value, ...) {
  cex = (value[[1]] - min(value[[1]]))/(max(value[[1]]) - min(value[[1]]))
  i = getI(

  circcos.genomicPoints(region, value, cex = cex, pch = 16, col = i, ...)
})

circcos.genomicTrackPlotRegion(bed_list, stack = TRUE, 
  panel.fun = function(region, value, ...) {
  cex = (value[[1]] - min(value[[1]]))/(max(value[[1]]) - min(value[[1]]))
  i = getI(
  circcos.genomicPoints(region, value, cex = cex, pch = 16, col = i, ...)
  cell.xlim = get.cell.meta.data("cell.xlim")
  circcos.lines(cell.xlim, c(i, i), lty = 2, col = "#00000040")
})
```r
bed = generateRandomBed(nr = 100, nc = 4)
circos.genomicTrackPlotRegion(bed, panel.fun = function(region, value, ...) {
  cex = (value[[1]] - min(value[[1]]))/(max(value[[1]]) - min(value[[1]]))
  circos.genomicPoints(region, value, cex = 0.5, pch = 16, col = 1:4, ...)
})
circos.genomicTrackPlotRegion(bed, stack = TRUE, panel.fun = function(region, value, ...) {
  cex = (value[[1]] - min(value[[1]]))/(max(value[[1]]) - min(value[[1]]))
  i = getI(...)
  circos.genomicPoints(region, value, cex = cex, pch = 16, col = i, ...)
  cell.xlim = get.cell.meta.data("cell.xlim")
  circos.lines(cell.xlim, c(i, i), lty = 2, col = "#00000040")
})
circos.clear()
```

---

circos.genomicPosTransformLines

*Add genomic position transformation lines between tracks*

**Description**

Add genomic position transformation lines between tracks

**Usage**

```r
circos.genomicPosTransformLines(
  data,
  track.height = 0.1,
  posTransform = NULL,
  horizontalLine = c("none", "top", "bottom", "both"),
  track.margin = c(0, 0),
  direction = c("inside", "outside"),
  col = "black",
  lwd = par("lwd"),
  lty = par("lty"),
  ...
)
```

**Arguments**

- **data**: A data frame containing genomic data
- **track.height**: Height of the track
- **posTransform**: Genomic position transformation function, see `posTransform.default` for an example.
- **horizontalLine**: Whether to draw horizontal lines which indicate region width
track.margin  Margin of tracks
direction     Type of the transformation. inside means position transformed track are located inside and outside means position transformed track are located outside.
col           Color of lines, can be length of one or nrow of data
lwd           Width of lines
lty           Style of lines
...           pass to circos.trackPlotRegion

Details

There is one representative situation when such position transformation needs to be applied. For example, there are two sets of regions in a chromosome in which regions in one set regions are quite densely to each other and regions in other set are far from others. Heatmap or text is going to be drawn on the next track. If there is no position transformation, heatmap or text for those dense regions would be overlapped and hard to identify, also ugly to visualize. Thus, a way to transform original positions to new positions would help for the visualization.

Examples

# There is no example
NULL
Arguments

data A bed-file-like data frame or a list of data frames
mode how to calculate the distance of two neighbouring regions, pass to `rainfallTransform`
ylim ylim for rainfall plot track. If `normalize_to_width` is FALSE, the value should correspond to log10(dist+1), and if `normalize_to_width` is TRUE, the value should correspond to log2(rel_dist).
col Color of points. It should be length of one. If data is a list, the length of col can also be the length of the list.
pch Style of points
cex Size of points

`normalize_to_width` 
If it is TRUE, the value is the relative distance divided by the width of the region.
...

Details

This is high-level graphical function, which mean, it will create a new track.
Rainfall plot can be used to visualize distribution of regions. On the plot, y-axis corresponds to the distance to neighbour regions (log-based). So if there is a drop-down on the plot, it means there is a cluster of regions at that area.
On the plot, y-axis are log10-transformed.

See Also


Examples

```r
## Not run:
load(system.file(package = "circlize", "extdata", "DMR.RData"))
# rainfall
circos.initializeWithIdeogram(plotType = c("axis", "labels"))

bed_list = list(DMR_hyper, DMR_hypo)
circos.genomicRainfall(bed_list, pch = 16, cex = 0.4, col = c("#FF000080", 
"#0000FF80"))
circos.genomicDensity(bed_list[[1]], col = c("#FF000080"), track.height = 0.1)
circos.genomicDensity(bed_list[[2]], col = c("#0000FF80"), track.height = 0.1)
circos.clear()

## End(Not run)
```
circos.genomicRect

Draw rectangle-like grid, specifically for genomic graphics

Description

Draw rectangle-like grid, specifically for genomic graphics

Usage

circos.genomicRect(
  region,
  value = NULL,
  ytop = NULL,
  ybottom = NULL,
  ytop.column = NULL,
  ybottom.column = NULL,
  sector.index = get.cell.meta.data("sector.index"),
  track.index = get.cell.meta.data("track.index"),
  posTransform = NULL,
  col = NA,
  border = "black",
  lty = par("lty"),
  ...
)

Arguments

region A data frame contains 2 column which correspond to start position and end position
value A data frame contains values and other information
ytop A vector or a single value indicating top position of rectangles
ybottom A vector or a single value indicating bottom position of rectangles
ytop.column If ytop is in value, the index of the column
ybottom.column If ybottom is in value, the index of the column
sector.index Pass to circos.rect
track.index Pass to circos.rect
posTransform Self-defined function to transform genomic positions, see posTransform.default for explanation
col The length of col can be either one or number of rows of region. Pass to circos.rect
border Settings are similar as col. Pass to circos.rect
lty Settings are similar as col. Pass to circos.rect
... Mysterious parameters
Details

The function is a low-level graphical function and usually is put in panel.fun when using \texttt{circos.genomicTrackPlotRegion}.

Examples

```r
## Not run:
############################
### rect matrix
circos.par("track.height" = 0.1, cell.padding = c(0, 0, 0, 0))
circos.initializeWithIdeogram(plotType = NULL)

bed = generateRandomBed(nr = 100, nc = 4)
circos.genomicTrackPlotRegion(bed, stack = TRUE, panel.fun = function(region, value, ...) {
  circos.genomicRect(region, value, col = sample(1:10, nrow(region), replace = TRUE),
                     border = NA, ...)
  i = getI(...)
  cell.xlim = get.cell.meta.data("cell xlim")
  #circos.lines(cell.xlim, c(i, i), lty = 2, col = "+00000040")
}, bg.border = NA)
circos.genomicPosTransformLines(bed, posTransform = posTransform.default,
                                 horizontalLine = "top")
circos.genomicTrackPlotRegion(bed, stack = TRUE, panel.fun = function(region, value, ...) {
  circos.genomicRect(region, value, col = sample(1:10, nrow(region), replace = TRUE),
                     border = NA, posTransform = posTransform.default, ...)
  i = getI(...)
  cell.xlim = get.cell.meta.data("cell_xlim")
  #circos.lines(cell.xlim, c(i, i), lty = 2, col = "+00000040")
}, bg.border = NA)
circos.genomicPosTransformLines(bed, posTransform = posTransform.default,
                                 direction = "outside", horizontalLine = "bottom")
circos.genomicTrackPlotRegion(bed, stack = TRUE, panel.fun = function(region, value, ...) {
  circos.genomicRect(region, value, col = sample(1:10, nrow(region), replace = TRUE),
                     border = NA, ...)
  i = getI(...)
  cell.xlim = get.cell.meta.data("cell_xlim")
  #circos.lines(cell.xlim, c(i, i), lty = 2, col = "+00000040")
}, bg.border = NA)
circos.clear()

############################
### rect from bed list
circos.par("track.height" = 0.1, cell.padding = c(0, 0, 0, 0))
circos.initializeWithIdeogram(plotType = NULL)

bed1 = generateRandomBed(nr = 100)
b bed2 = generateRandomBed(nr = 100)
bed_list = list(bed1, bed2)
```

circos.genomicText

Draw text in a cell, specifically for genomic graphics

description

Draw text in a cell, specifically for genomic graphics

Usage

circos.genomicText(
    region,
    value = NULL,
    y = NULL,
    labels = NULL,
labels.column = NULL,
numeric.column = NULL,
sector.index = get.cell.meta.data("sector.index"),
track.index = get.cell.meta.data("track.index"),
posTransform = NULL,
direction = NULL,
facing = "inside",
niceFacing = FALSE,
adj = par("adj"),
cex = 1,
col = "black",
font = par("font"),
padding = 0,
extend = 0,
align_to = "region",
...)

Arguments

region          A data frame contains 2 column which correspond to start position and end position
value           A data frame contains values and other information
y               A vector or a single value indicating position of text.
labels          Labels of text corresponding to each genomic positions
labels.column   If labels are in value, index of column in value
numeric.column  Which column in value data frame should be taken as y-value. If it is not defined, only the first numeric columns in value will be taken.
sector.index    Pass to circos.rect
track.index     Pass to circos.rect
posTransform    Self-defined function to transform genomic positions, see posTransform.default for explanation
facing          Passing to circos.text. Settings are similar as col
niceFacing      Should the facing of text be adjusted to fit human eyes?
direction       Deprecated, use facing instead.
adj             Pass to circos.text. Settings are similar as col
cex             Pass to circos.text. Settings are similar as col
col             Pass to circos.text. The length of col can be either one or number of rows of region.
font            Pass to circos.text. Settings are similar as col
padding         pass to posTransform if it is set as posTransform.text
extend          pass to posTransform if it is set as posTransform.text
align_to        pass to posTransform if it is set as posTransform.text
...             Mysterious parameters
Details

The function is a low-level graphical function and usually is put in panel.fun when using `circos.genomicTrackPlotRegion`.

Examples

```r
circos.par("track.height" = 0.1, cell.padding = c(0, 0, 0, 0))
circos.initializeWithIdeogram(plotType = NULL)

bed = generateRandomBed(nr = 20)

circos.genomicTrackPlotRegion(bed, ylim = c(0, 1), panel.fun = function(region, value, ...) {
  circos.genomicText(region, value, y = 0.5, labels = "text", ...)
})

bed = cbind(bed, sample(letters, nrow(bed), replace = TRUE))
circos.genomicTrackPlotRegion(bed, panel.fun = function(region, value, ...) {
  circos.genomicText(region, value, labels.column = 2, ...)
})
circos.clear()
```

circos.genomicTrack

Create a track for genomic graphics

Description

Create a track for genomic graphics

Usage

```r
circos.genomicTrack(...)
```

Arguments

... pass to `circos.genomicTrackPlotRegion`

Details

shortcut function of `circos.genomicTrackPlotRegion`.

Examples

```r
# There is no example
NULL
```
Create a track for genomic graphics

**Usage**

```r
circos.genomicTrackPlotRegion(
    data = NULL,
    ylim = NULL,
    stack = FALSE,
    numeric.column = NULL,
    jitter = 0,
    panel.fun = function(region, value, ...) {NULL},
    ...
)
```

**Arguments**

- `data` A bed-file-like data frame or a list of data frames
- `ylim` If it is `NULL`, the value will be calculated from data. If `stack` is set to `TRUE`, this value is ignored.
- `stack` Whether to plot in a "stack" mode.
- `numeric.column` Columns of numeric values in `data` that will be used for plotting. If `data` is a data frame list, `numeric.column` should be either length of one or length of `data`. If value of `numeric.column` is not set, its value will depend on the structure of `data`. If `data` is a data frame, the default value for `numeric.column` is all the numeric column starting from the fourth column. If `data` is a list of data frame, the default value for `numeric.column` is a vector which have the same length as `data` and the value in default `numeric.column` is the index of the first numeric column in corresponding data frame.
- `jitter` Numeric. Only works for adding points in `circos.genomicTrackPlotRegion` under stack mode.
- `panel.fun` Self-defined function which will be applied on each sector. Please note it is different from that in `circos.trackPlotRegion`. In this function, there are two arguments (region and value) plus `...`. In them, region is a two-column data frame with start positions and end positions in current genomic category (e.g. chromosome). value is a data frame which is derived from data but excluding the first three columns. Rows in value correspond to rows in region. ... is mandatory and is used to pass internal parameters to other functions. The definition of value will be different according to different input data (data frame or list of data frame) and different settings (stacked or not), please refer to 'details' section and vignettes to detailed explanation.

... Pass to `circos.trackPlotRegion`. 
Details

Similar as `circos.trackPlotRegion`, users can add customized graphics by `panel.fun`, but the behaviour of `panel.fun` will change depending on users’ input data and stack setting.

When data is a single data frame, `region` in `panel.fun` is a data frame containing the second and third column in data in 'current' genomic category (e.g. current chromosome). `value` is also a data frame containing columns in data excluding the first three columns.

When data is a list containing data frames, `panel.fun` will be applied iteratively on each data frame, thus, `region` is extracted from the data frame which is in the current iteration. For example, if data contains two data frames, `panel.fun` will be applied with the first data frame in current chromosome and then applied with the second data frame in the same chromosome.

If `stack` is set to `TRUE`, `ylim` will be re-defined. In stack mode, the y-axis will be split into several part with equal height and graphics will be drawn on each 'horizontal' lines (y = 1, 2, ...). In this case:

When data is a single data frame containing one or more numeric columns, each numeric column defined in `numeric.column` will be treated as a single unit. `ylim` is re-defined to `c(0.5, n+0.5)` in which `n` is number of numeric columns. `panel.fun` will be applied iteratively on each numeric column. In each iteration, in `panel.fun`, `region` is still the genomic regions in current genomic category, but `value` contains current numeric column plus all non-numeric columns. Under stack mode, in `panel.fun`, all low-level genomic graphical functions will draw on the 'horizontal line' `y = i` in which `i` is the index of current numeric column and the value of `i` can be obtained by `getI`

When data is a list containing data frames, each data frame will be treated as a single unit. The situation is quite similar as described in previous paragraph. `ylim` is re-defined to `c(0.5, n+0.5)` in which `n` is number of data frames. `panel.fun` will be applied iteratively on each data frame. In each iteration, in `panel.fun`, `region` is still the genomic regions in current genomic category, and `value` contains columns in current data frame excluding the first three columns. Under stack mode, in `panel.fun`, all low-level genomic graphical functions will draw on the 'horizontal line' `y = i` in which `i` is the index of current data frame.

Being different from `panel.fun` in `circos.trackPlotRegion`, there should be an additional argument `...` in `panel.fun`. This additional argument is used to pass hidden values to low-level graphical functions. So if you are using functions like `circos.genomicPoints`, you should also add `...` as an additional argument into `circos.genomicPoints`.

See Also


Examples

```r
# There is no example
NULL
```
circos.info

Get information of the circular plot

Description

Get information of the circular plot

Usage

circos.info(sector.index = NULL, track.index = NULL, plot = FALSE)

Arguments

sector.index  Which sectors you want to look at? It can be a vector.
track.index  Which tracks you want to look at? It can be a vector.
plot  Whether to add information on the plot

Details

It tells you the basic parameters for sectors/tracks/cells. If both sector.index and track.index are set to NULL, the function would print index for all sectors and all tracks. If sector.index and/or track.index are set, the function would print xlim, ylim, cell.xlim, cell ylim, xplot, yplot, track.margin and cell.padding for every cell in specified sectors and tracks. Also, the function will print index of your current sector and current track.

If plot is set to TRUE, the function will plot the index of the sector and the track for each cell on the figure.

See Also


Examples

# There is no example
NULL
**circos.initialize** *Initialize the circular layout*

**Description**

Initialize the circular layout

**Usage**

```r
circos.initialize(
  factors,
  x = NULL,
  xlim = NULL,
  sector.width = NULL)
```

**Arguments**

- `factors`: A `factor` variable or a character vector which represent data categories
- `x`: Data on x-axes, a vector
- `xlim`: Ranges for values on x-axes, see "details" section for explanation of the format
- `sector.width`: Width for each sector. The length of the vector should be either 1 which means all sectors have same width or as same as the number of sectors. Values for the vector are relative, and they will be scaled by dividing their summation. By default, it is `NULL` which means the width of sectors correspond to the data range in sectors.

**Details**

The function allocates the sectors according to the values on x-axis. The number of sectors are determined by the `factors` and the order of sectors are determined by the levels of factors. In this function, the start and end position for each sector on the circle (measured by degree) are calculated according to the values on x-axis or by `xlim`.

If `x` is set, the length of `x` must be equal to the length of `factors`. Then the data range for each sector are calculated from `x` by splitting `factors`.

If `xlim` is set, it should be a vector containing two numbers or a matrix with 2 columns. If `xlim` is a 2-element vector, it means all sector share the same `xlim`. If `xlim` is a 2-column matrix, the number of rows should be equal to the number of categories identified by `factors`, then each row of `xlim` corresponds to the data range for each sector and the order of rows is corresponding to the order of levels of `factors`. If `xlim` is a matrix for which row names cover all sector names, `xlim` is automatically adjusted.

Normally, width of sectors will be calculated internally according to the data range in sectors. But you can still set the width manually. However, it is not always a good idea to change the default sector width since the width can reflect the range of data in sectors. However, in some cases, it is useful to manually set the width such as you want to zoom some part of the sectors.

The function finally calls `plot` with enforcing aspect ratio to be 1 and be ready for adding graphics.
circos.initializeWithIdeogram

Initialize the circular layout with an ideogram

Description

Initialize the circular layout with an ideogram

Usage

```r
circos.initializeWithIdeogram(
  cytoband = system.file(package = "circlize", "extdata", "cytoBand.txt"),
  species = NULL,
  sort.chr = TRUE,
  chromosome.index = usable_chromosomes(species),
  major.by = NULL,
  plotType = c("ideogram", "axis", "labels"),
  track.height = NULL,
  ideogram.height = convert_height(2, "mm"),
  ...
)
```

Arguments

- `cytoband`: A path of the cytoband file or a data frame that already contains cytoband data. By default it is cytoband for hg19. Pass to `read.cytoband`
- `species`: Abbreviations of species. e.g. hg19 for human, mm10 for mouse. If this value is specified, the function will download cytoBand.txt.gz from UCSC website automatically. If there is no cytoband for user’s species, it will keep on trying to download chromInfo file. Pass to `read.cytoband` or `read.chromInfo`
- `chromosome.index`: subset of chromosomes, also used to reorder chromosomes.
- `sort.chr`: Whether chromosome names should be sorted (first sort by numbers then by letters). If chromosome.index is set, this argument is enforced to FALSE
- `major.by`: Increment of major ticks. Pass to `circos.genomicInitialize`
- `plotType`: Which tracks should be drawn. ideogram for ideogram rectangle, axis for genomic axis and labels for chromosome names. If there is no ideogram for specified species, ideogram will be enforced to be excluded. If it is set to NULL, the function just initialize the plot but draw nothing.
The function will initialize the circular plot in which each sector corresponds to a chromosome. You can control the order of chromosomes by chromosome.index or by sort.chr, or by setting a special format of cytoband (please refer to read.cytoband to find out how to control a proper cytoband).

The function finally pass data to circos.genomicInitialize to initialize the circular plot. The style of ideogram is almost fixed, but you can customize it with your self-defined code. Refer to vignette for demonstration.

See Also


Examples

```r
## Not run:
circos.initializeWithIdeogram()

cytoband.file = system.file(package = "circlize", "extdata", "cytoBand.txt")
circos.initializeWithIdeogram(cytoband.file)

cytaband.df = read.table(cytoband.file, colClasses = c("character", "numeric", "numeric", "character", "character"), sep = "\t")
circos.initializeWithIdeogram(cytoband.df)

circos.initializeWithIdeogram(species = "hg18")

circos.initializeWithIdeogram(species = "mm10")

circos.initializeWithIdeogram(chromosome.index = c("chr1", "chr2"))

cytaband = read.table(cytoband.file, colClasses = c("character", "numeric", "numeric", "character", "character"), sep = "\t")
circos.initializeWithIdeogram(cytoband, sort.chr = FALSE)


cytaband[[1]] = factor(cytoband[[1]], levels = paste0("chr", c(22:1, "X", "Y")))
circos.initializeWithIdeogram(cytoband, sort.chr = FALSE)

cytaband = read.table(cytoband.file, colClasses = c("character", "numeric", "numeric", "character", "character"), sep = "\t")
circos.initializeWithIdeogram(cytoband, sort.chr = TRUE)

circos.initializeWithIdeogram(plotType = c("axis", "labels"))
```
circos.initializeWithIdeogram(plotType = NULL)

circos.par("start.degree" = 90)
circos.initializeWithIdeogram()
circos.clear()

circos.par("gap.degree" = rep(c(2, 4), 12))
circos.initializeWithIdeogram()
circos.clear()

## End(Not run)

circos.lines

Add lines to the plotting region

Description
Add lines to the plotting region

Usage

circos.lines(
  x, y,
  sector.index = get.cell.meta.data("sector.index"),
  track.index = get.cell.meta.data("track.index"),
  col = ifelse(area, "grey", par("col")),
  lwd = par("lwd"),
  lty = par("lty"),
  type = "l",
  straight = FALSE,
  area = FALSE,
  area.baseline = NULL,
  border = "black",
  baseline = "bottom",
  pt.col = par("col"),
  cex = par("cex"),
  pch = par("pch"))

Arguments

x        Data points on x-axis, measured in "current" data coordinate
y        Data points on y-axis, measured in "current" data coordinate
sector.index  Index for the sector
track.index  Index for the track
col        Line color
lwd        line width
circos.lines  

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>lty</td>
<td>line style</td>
</tr>
<tr>
<td>type</td>
<td>line type, similar as type argument in <code>lines</code>, but only in <code>c(&quot;l&quot;,&quot;o&quot;,&quot;h&quot;,&quot;s&quot;)</code></td>
</tr>
<tr>
<td>straight</td>
<td>whether draw straight lines between points.</td>
</tr>
<tr>
<td>area</td>
<td>whether to fill the area below the lines. If it is set to <code>TRUE</code>, <code>col</code> controls the filled color in the area and <code>border</code> controls color of the line.</td>
</tr>
<tr>
<td>area.baseline</td>
<td>deprecated, use <code>baseline</code> instead.</td>
</tr>
<tr>
<td>baseline</td>
<td>the base line to draw areas. By default it is the minimal of y-range (bottom). It can be a string or a number. If a string, it should be one of <code>bottom</code> and <code>top</code>. This argument also works if <code>type</code> is set to <code>h</code>.</td>
</tr>
<tr>
<td>border</td>
<td>color for border of the area</td>
</tr>
<tr>
<td>pt.col</td>
<td>if type is &quot;o&quot;, point color</td>
</tr>
<tr>
<td>cex</td>
<td>if type is &quot;o&quot;, point size</td>
</tr>
<tr>
<td>pch</td>
<td>if type is &quot;o&quot;, point type</td>
</tr>
</tbody>
</table>

**Details**

Normally, straight lines in the Cartesian coordinate have to be transformed into curves in the circular layout. But if you do not want to do such transformation you can use this function just drawing straight lines between points by setting `straight` to `TRUE`.

Drawing areas below lines can help to identify the direction of y-axis in cells (since it is a circle). This can be done by specifying `area` to `TRUE`.

**See Also**

- `factors` = letters[1:9]  
- `circos.par(points.overflow.warning = FALSE)`  
- `circos.initialize(factors = factors, xlim = c(0, 10))`  
- `circos.trackPlotRegion(factors = factors, ylim = c(0, 10), track.height = 0.5)`  
- `circos.lines(sort(runif(10)*10), runif(10)*8, sector.index = "a")`  
- `circos.text(5, 9, "type = 'l'", sector.index = "a", facing = "outside")`  
- `circos.lines(sort(runif(10)*10), runif(10)*8, sector.index = "b", type = "o")`  
- `circos.text(5, 9, "type = 'o'", sector.index = "b", facing = "outside")`  
- `circos.lines(sort(runif(10)*10), runif(10)*8, sector.index = "c", type = "h")`  
- `circos.text(5, 9, "type = 'h'", sector.index = "c", facing = "outside")`  
- `circos.lines(sort(runif(10)*10), runif(10)*8, sector.index = "d", type = "h", baseline = 5)`  
- `circos.text(5, 9, "type = 'h'", baseline = 5, sector.index = "d", facing = "outside")`  
- `circos.lines(sort(runif(10)*10), runif(10)*8, sector.index = "e", type = "s")`  
- `circos.text(5, 9, "type = 's'", sector.index = "e", facing = "outside")`  
- `circos.lines(sort(runif(10)*10), runif(10)*8, sector.index = "f", area = TRUE)`  
- `circos.text(5, 9, "type = 'l'", area = TRUE, sector.index = "f")`  
- `circos.lines(sort(runif(10)*10), runif(10)*8, sector.index = "g", type = "o", area = TRUE)`  
- `circos.text(5, 9, "type = 'o'", area = TRUE, sector.index = "g")`  
- `circos.lines(sort(runif(10)*10), runif(10)*8, sector.index = "h", type = "s", area = TRUE)`  
- `circos.text(5, 9, "type = 's'", area = TRUE, sector.index = "h")`  
- `circos.lines(sort(runif(10)*10), runif(10)*8, sector.index = "i", area = TRUE, baseline = "top")`  
- `circos.text(5, 9, "type = 'l'", area = TRUE, baseline = "top", sector.index = "i")`  
- `circos.clear()`
circos.link

Draw links between points or/and intervals

Description

Draw links between points or/and intervals

Usage

circos.link(
  sector.index1,
  point1,
  sector.index2,
  point2,
  rou = get_most_inside_radius(),
  rou1 = rou,
  rou2 = rou,
  h = NULL,
  h.ratio = 0.5,
  w = 1,
  h2 = h,
  w2 = w,
  col = "black",
  lwd = par("lwd"),
  lty = par("lty"),
  border = col,
  directional = 0,
  arr.length = ifelse(arr.type == "big.arrow", 0.02, 0.4),
  arr.width = arr.length/2,
  arr.type = "triangle",
  arr.lty = lty,
  arr.lwd = lwd,
  arr.col = col)

Arguments

- **sector.index1**: Index for the first sector where one link end locates
- **point1**: A single value or a numeric vector of length 2. If it is a 2-elements vector, then the link would be a belt/ribbon.
- **sector.index2**: Index for the other sector where the other link end locates

Examples

# There is no example
NULL
point2 A single value or a numeric vector of length 2. If it is a 2-elements vector, then the link would be a belt/ribbon.

rou The position of the the link ends (if rou1 and rou2 are not set). It is the percentage of the radius of the unit circle. By default its value is the position of bottom margin of the most inner track.

rou1 The position of end 1 of the link.

rou2 The position of end 2 of the link.

h Height of the link, measured as percent to the radius to the unit circle. By default it is automatically inferred.

h.ratio systematically change the link height. The value is between 0 and 1.

w Since the link is a Bezier curve, it controls the shape of Bezier curve.

h2 Height of the bottom edge of the link if it is a ribbon.

w2 Shape of the bottom edge of the link if it is a ribbon.

col Color of the link. If the link is a ribbon, then it is the filled color for the ribbon.

lwd Line (or border) width

lty Line (or border) style

border If the link is a ribbon, then it is the color for the ribbon border.

directional 0 for no direction, 1 for direction from point1 to point2, -1 for direction from point2 to point1. 2 for two directional. The direction is important when arrow heads are added.

arr.width Width of the arrows, pass to Arrowhead.

arr.type Type of the arrows, pass to Arrowhead. Default value is triangle. There is an additional option big.arrow.

arr.length Length of the arrows, measured in 'cm', pass to Arrowhead. If arr.type is set to big.arrow, the value is percent to the radius of the unit circle.

arr.col Color of the arrows, pass to Arrowhead.

arr.lwd Line width of arrows, pass to Arrowhead.

arr.lty Line type of arrows, pass to Arrowhead.

Details

Links are implemented as quadratic Bezier curves (https://en.wikipedia.org/wiki/B%C3%A9zier_curve#Rational_B.C3.A9zier_curves).

Drawing links does not create any track. So you can think it is independent of the tracks.

By default you only need to set sector.index1, point1, sector.index2 and point2. The links would look nice.

Please refer to the vignette for detailed explanation.

See Also

Examples

    # There is no example
    NULL

Description

Nested zooming with two circular plots

Usage

    circos.nested(
      f1,
      f2,
      correspondance,
      connection_height = convert_height(5, "mm"),
      connection_col = NA,
      connection_border = "black",
      connection_lty = par("lty"),
      connection_lwd = par("lwd"),
      adjust_start_degree = TRUE)

Arguments

f1 a self-defined function for making the first circular plot. The function should have no argument.
f2 a self-defined function for making the second circular plot. The function should have no argument.
correspondance a six-column data frame which contains correspondance between the coordinates in two circular plots
connection_height the height of the connection track, measured as the percent to the radius of the unit circle. The value can be specified by uh or convert_height with absolute units.
connection_col filled color of the connection track. The value can be a vector with same length as number of rows of correspondance
connection_border border color of the connection track.
connection_lty line style of the connection track borders
connection_lwd line width of the connection track borders
adjust_start_degree If circos.par(start.degree = ...) is not set in f2(), the start degree for the second circular plot will be adjusted to make the distance of sectors between the two plots to the minimal.
Details

The function visualizes zoomings by combining two circular plots into one page where one is the normal circular plot and the other one only contains regions that need to be zoomed. This function automatically arranges the two plots to make it easy to correspond between the original and the zoomed sectors.

Since the function needs to know the information of the two circular plots, please do not call `circos.clear` in either `f1()` or `f2()`. It will be called internally in `circos.nested`.

If `adjust_start_degree` is set to `TRUE`, `start_degree` should not be set in `f2()`. Also `canvas.xlim` and `canvas ylim` are reset in `f2()`, they should not be set in `f2()` either.

Author(s)

Zuguang Gu <z.gu@dkfz.de>

See Also


Examples

```r
# There is no example
NULL
```

---

circos.par  Parameters for the circular layout

Description

Parameters for the circular layout

Usage

```r
circos.par(..., RESET = FALSE, READ.ONLY = NULL, LOCAL = FALSE, ADD = FALSE)
```

Arguments

- `...`: Arguments for the parameters, see "details" section
- `RESET`: reset to default values
- `READ.ONLY`: please ignore
- `LOCAL`: please ignore
- `ADD`: please ignore
Details

Global parameters for the circular layout. Currently supported parameters are:

start.degree The starting degree from which the circle begins to draw. Note this degree is measured in the standard polar coordinate which means it is always reverse-clockwise.

gap.degree Gap between two neighbour sectors. It can be a single value or a vector. If it is a vector, the first value corresponds to the gap after the first sector.

gap.after identical to gap.degree option, but a more understandable name. Modifying this option will also affect gap.degree.

track.margin Like margin in Cascading Style Sheets (CSS), it is the blank area out of the plotting region, also outside of the borders. Since left and right margin are controlled by gap.degree, only bottom and top margin need to be set. And all cells in a same track share the same margins, and that’s why this parameter is called track.margin. The value for the track.margin is the percentage according to the radius of the unit circle. convert_height can be used to set to an absolute unit (e.g cm/inche).

unit.circle.segments Since curves are simulated by a series of straight lines, this parameter controls the amount of segments to represent a curve. The minimal length of the line segmentation is the length of the unit circle (2pi) divided by unit.circle.segments. More segments means better approximation for the curves while larger size if you generate figures as PDF format.

cell.padding Padding of the cell. Like padding in Cascading Style Sheets (CSS), it is the blank area around the plotting regions, but within the borders. The parameter has four values, which controls the bottom, left, top and right paddings respectively. The first and the third padding values are the percentages according to the radius of the unit circle and the second and fourth values are degrees. Similar as track.margin option, the first and the third value can be set by convert_height to an absolute unit.

track.height The default height of tracks. It is the percentage according to the radius of the unit circle. The height includes the top and bottom cell paddings but not the margins. convert_height can be used to set the height to an absolute unit.

points.overflow.warning Since each cell is in fact not a real plotting region but only an ordinary rectangle, it does not eliminate points that are plotted out of the region. So if some points are out of the plotting region, circlize would continue drawing the points and printing warnings. In some cases, draw something out of the plotting region is useful, such as draw some legend or text. Set this value to FALSE to turn off the warnings.

canvas.xlim The coordinate for the canvas. Because circlize draws everything (or almost everything) inside the unit circle, the default canvas.xlim and canvas.ylim for the canvas would be all c(-1,1). However, you can set it to a more broad interval if you want to draw other things out of the circle. By choosing proper canvas.xlim and canvas.ylim, you can draw part of the circle. E.g. setting canvas.xlim to c(0,1) and canvas.ylim to c(0,1) would only draw circle in the region of (0, pi/2).

canvas.ylim The coordinate for the canvas. By default it is c(-1,1)

clock.wise The direction for adding sectors. Default is TRUE.

Similar as par, you can get the parameter values by specifying the names of parameters and you can set the parameter values by specifying a named list which contains the new values.


gap\_degree, start\_degree, canvas\_xlim, canvas\_ylim and clock\_wise only be set before the initialization of the circular layout (i.e. before calling \texttt{circos.initialize}) because these values will not be changed after adding sectors on the circle. The left and right padding for \texttt{cell.padding} will also be ignored after the initialization because all cells in a sector would share the same left and right paddings.

**Examples**

```r
# There is no example
NULL
```

---

**circos.points**  
*Add points to a plotting region*

**Description**

Add points to a plotting region

**Usage**

```r
circos.points(
  x, y,
  sector.index = get.cell.meta.data("sector.index"),
  track.index = get.cell.meta.data("track.index"),
  pch = par("pch"),
  col = par("col"),
  cex = par("cex"),
  bg = par("bg"))
```

**Arguments**

- **x**  
  Data points on x-axis, measured in "current" data coordinate
- **y**  
  Data points on y-axis, measured in "current" data coordinate
- **sector.index**  
  Index for the sector
- **track.index**  
  Index for the track
- **pch**  
  Point type
- **col**  
  Point color
- **cex**  
  Point size
- **bg**  
  Background of points
Details

This function can only add points in one specified cell. Pretending a low-level plotting function, it can only be applied in plotting region which has been created.

You can think the function similar as the normal points function, just adding points in the circular plotting region. The position of cell is identified by sector.index and track.index, if they are not specified, they are in 'current' sector and 'current' track.

Data points out of the plotting region will also be added, but with warning messages.

Other graphics parameters which are available in the function are pch, col and cex which have same meaning as those in the par.

It is recommended to use circos.points inside panel.fun in circos.trackPlotRegion so that it draws points directly on "current" cell.

See Also


Examples

```r
circos.initialize(letters[1:8], xlim = c(0, 1))
circos.track(ylim = c(0, 1), panel.fun = function(x, y) {
  circos.points(runif(10), runif(10))
})
circos.points(runif(10), runif(10), sector.index = "e", pch = 16, col = "red")
circos.clear()
```

circos.polygon

Description

Draw polygon

Usage

```r
circos.polygon(
  x, y,
  sector.index = get.cell.meta.data("sector.index"),
  track.index = get.cell.meta.data("track.index"),
  ...
)
```

Arguments

- **x**  
  Data points on x-axis
- **y**  
  Data points on y-axis
- **sector.index**  
  Index for the sector
- **track.index**  
  Index for the track
- **...**  
  pass to polygon
Details

similar as `polygon`.

Note: start point should overlap with the end point,

Examples

```r
def set.seed(123)
  factors = letters[1:4]
circos.initialize(factors, xlim = c(0, 1))
circos.trackPlotRegion(ylim = c(-3, 3), track.height = 0.4, panel.fun = function(x, y) {
  x1 = runif(20)
y1 = x1 + rnorm(20)
or = order(x1)
x1 = x1[or]
y1 = y1[or]
loess.fit = loess(y1 ~ x1)
loess.predict = predict(loess.fit, x1, se = TRUE)
d1 = c(x1, rev(x1))
d2 = c(loess.predict$fit + loess.predict$se.fit,
     rev(loess.predict$fit - loess.predict$se.fit))
circos.polygon(d1, d2, col = "CCCCCC", border = NA)
circos.points(x1, y1, cex = 0.5)
circos.lines(x1, loess.predict$fit)
})
circos.clear()
```

---

circos.raster

Add raster images

Description

Add raster images

Usage

circos.raster(
  image, x, y,
  width, height,
  facing = c("inside", "outside", "reverse.clockwise", "clockwise",
            "downward", "bending.inside", "bending.outside"),
  niceFacing = FALSE,
  sector.index = get.cell.meta.data("sector.index"),
  track.index = get.cell.meta.data("track.index"),
  scaling = 1)
Arguments

- **image**: a raster object, or an object that can be converted by `as.raster`
- **x**: position of the center of the raster image, measured in the data coordinate in the cell
- **y**: position of the center of the raster image, measured in the data coordinate in the cell
- **width**: width of the raster image. When facing is one of "inside", "outside", "clockwise" and "reverse.clockwise", the image should have absolute size where the value of width should be specified like 20mm, 1cm or 0.5inches. When facing is one of bending.inside and bending.outside, the value of width is measured in the data coordinate in the cell.
- **height**: height of the raster image. Same format as width. If the value of height is omit, default height is calculated by taking the aspect ratio of the original image. But when facing is one of bending.inside and bending.outside, height is mandatory to set.
- **facing**: facing of the raster image
- **niceFacing**: facing of text. Please refer to vignette for different settings
- **sector.index**: index for the sector
- **track.index**: index for the track
- **scaling**: scaling factor to resize the raster image.

Author(s)

Zuguang Gu <z.gu@dkfz.de>

Examples

```R
require(png)
image = system.file("extdata", "Rlogo.png", package = "circlize")
image = as.raster(readPNG(image))
circos.initialize(letters[1:8], xlim = c(0, 1))
circos.track(ylim = c(0, 1), panel.fun = function(x, y) {
  circos.raster(image, CELL_META$xcenter, CELL_META$ycenter, width = "2cm",
                facing = "inside", niceFacing = TRUE)
})
circos.clear()

if(FALSE) {
  # NOTE: following takes quite a long time to run
  load(system.file("extdata", "doodle.RData", package = "circlize"))
circos.par("cell.padding" = c(0, 0, 0, 0))
circos.initialize(letters[1:16], xlim = c(0, 1))
circos.track(ylim = c(0, 1), panel.fun = function(x, y) {
  img = img_list[[CELL_META$sector.numeric.index]]
circos.raster(img, CELL_META$xcenter, CELL_META$ycenter, width = 1,
               height = 1, facing = "bending.inside")
  }, track.height = 0.25, bg.border = NA)
```

```r
circos.track(ylim = c(0, 1), panel.fun = function(x, y) {
  img = img_list[[CELL_META$sector.numeric.index + 16]]
  circos.raster(img, CELL_META$xcenter, CELL_META$ycenter, width = 1,
               height = 1, facing = "bending.inside")
}, track.height = 0.25, bg.border = NA)
circos.clear()
}
circos.rect
```

---

**circos.rect**  
*Draw rectangle-like grid*

**Description**

Draw rectangle-like grid

**Usage**

```r
circos.rect(
  xleft, ybottom, xright, ytop,
  sector.index = get.cell.meta.data("sector.index"),
  track.index = get.cell.meta.data("track.index"),
  rot = 0,
  ...
)
```

**Arguments**

- `xleft`  
  x for the left bottom points

- `ybottom`  
  y for the left bottom points

- `xright`  
  x for the right top points

- `ytop`  
  y for the right top points

- `sector.index`  
  Index for the sector

- `track.index`  
  Index for the track

- `rot`  
  Rotation of the rectangles. The value is measured clockwise in degree. Rotation is relative to the center of the rectangles.

- `...`  
  pass to `polygon`

**Details**

The name for this function is **circos.rect** because if you imagine the plotting region as Cartesian coordinate, then it is rectangle. In the polar coordinate, the up and bottom edge become two arcs.

This function can be vectorized.
Examples

circos.initialize(fa = c("a", "b", "c", "d"), xlim = c(0, 10))
circos.track(ylim = c(0, 10), panel.fun = function(x, y) {
  for(rot in seq(0, 360, by = 30)) {
    circos.rect(2, 2, 6, 6, rot = rot)
  }
}, track.height = 0.5)

circos.segments  Draw segments through pairwise of points

Description

Draw segments through pairwise of points

Usage

circos.segments(
  x0, y0, x1, y1,
  sector.index = get.cell.meta.data("sector.index"),
  track.index = get.cell.meta.data("track.index"),
  straight = FALSE,
  col = par("col"),
  lwd = par("lwd"),
  lty = par("lty"),
  ...
)

Arguments

x0        x coordinates for starting points
y0        y coordinates for ending points
x1        x coordinates for starting points
y1        y coordinates for ending points
sector.index Index for the sector
track.index Index for the track
straight  whether the segment is a straight line
col       color of the segments
lwd       line width of the segments
lty       line type of the segments
...       pass to lines

Examples

# There is no example
NULL
**circos.text**  

**Draw text in a cell**

**Description**

Draw text in a cell

**Usage**

```r
circos.text(
  x, y,
  labels,
  sector.index = get.cell.meta.data("sector.index"),
  track.index = get.cell.meta.data("track.index"),
  direction = NULL,
  facing = c("inside", "outside", "reverse.clockwise", "clockwise",
             "downward", "bending", "bending.inside", "bending.outside"),
  niceFacing = FALSE,
  adj = par("adj"),
  cex = 1,
  col = par("col"),
  font = par("font"),
  ...
)
```

**Arguments**

- `x`  
  Data points on x-axis
- `y`  
  Data points on y-axis
- `labels`  
  Labels for each points
- `sector.index`  
  Index for the sector
- `track.index`  
  Index for the track
- `direction`  
  deprecated, use `facing` instead.
- `facing`  
  Facing of text. Please refer to vignette for different settings
- `niceFacing`  
  Should the facing of text be adjusted to fit human eyes?
- `adj`  
  offset for text. By default the text position adjustment is either horizontal or vertical in the canvas coordinate system. The "circular horizontal" offset can be set as a value in degree unit and the value should be wrapped by `degree`
- `...`  
  Pass to `text`
- `cex`  
  Font size
- `col`  
  Font color
- `font`  
  Font style

**Details**

The function is similar to `text`. All you need to note is the `facing` settings.
See Also


Examples

```r
factors = letters[1:4]
circos.par(points.overflow.warning = FALSE)
circos.initialize(factors = factors, xlim = c(0, 10))
circos.trackPlotRegion(factors = factors, ylim = c(0, 10),
  track.height = 0.5, panel.fun = function(x, y) {
    circos.text(3, 1, "inside", facing = "inside", cex = 0.8)
    circos.text(7, 1, "outside", facing = "outside", cex = 0.8)
    circos.text(0, 5, "reverse.clockwise", facing = "reverse.clockwise",
               adj = c(0.5, 0), cex = 0.8)
    circos.text(10, 5, "clockwise", facing = "clockwise", adj = c(0.5, 0),
               cex = 0.8)
    circos.text(5, 5, "downward", facing = "downward", cex = 0.8)
    circos.text(3, 9, "====bending.inside====", facing = "bending.inside",
               cex = 0.8)
    circos.text(7, 9, "====bending.outside====", facing = "bending.outside",
               cex = 0.8)
  })
circos.clear()
```

---

circos.track

Create plotting regions for a whole track

Description

Create plotting regions for a whole track

Usage

```r
circos.track(...)```

Arguments

... pass to `circos.trackPlotRegion`

Details

Shortcut function of `circos.trackPlotRegion`.

Examples

```r
# There is no example
NULL
```
**circos.trackHist**  
*Draw histogram in cells among a whole track*

**Description**

Draw histogram in cells among a whole track

**Usage**

```r
circos.trackHist(
    factors,
    x,
    track.height = circos.par("track.height"),
    track.index = NULL,
    force.ylim = TRUE,
    col = ifelse(draw.density, "black", NA),
    border = "black",
    lty = par("lty"),
    lwd = par("lwd"),
    bg.col = NA,
    bg.border = "black",
    bg.lty = par("lty"),
    bg.lwd = par("lwd"),
    breaks = "Sturges",
    include.lowest = TRUE,
    right = TRUE,
    draw.density = FALSE,
    bin.size = NULL,
    area = FALSE)
```

**Arguments**

- **factors**  
  Factors which represent the categories of data
- **x**  
  Data on the x-axis
- **track.index**  
  Index for the track which is going to be updated. Setting it to `NULL` means creating the plotting regions in the next newest track.
- **track.height**  
  Height of the track. It is the percentage to the radius of the unit circle. If to update a track, this argument is disabled.
- **force.ylim**  
  Whether to force all cells in the track to share the same ylim. Btw, ylim is calculated automatically.
- **col**  
  Filled color for histogram
- **border**  
  Border color for histogram
- **lty**  
  Line style for histogram
- **lwd**  
  Line width for histogram
circos.trackLines

Add lines to the plotting regions in a same track

Description

Add lines to the plotting regions in a same track

### bg.col
Background color for the plotting regions

### bg.border
Color for the border of the plotting regions

### bg.lty
Line style for the border of the plotting regions

### bg.lwd
Line width for the border of the plotting regions

### breaks
see hist

### include.lowest
see hist

### right
see hist

### draw.density
whether draw density lines instead of histogram bars.

### area
whether to fill the area below the density lines. If it is set to TRUE, col controls the filled color in the area and border controls color of the line.

### bin.size
size of the bins of the histogram

Details

It draw histogram in cells among a whole track. It is also an example to show how to add self-defined high-level graphics by this package.

See Also


Examples

```r
## Not run:
x = rnorm(1600)
factors = sample(letters[1:16], 1600, replace = TRUE)
circos.initialize(factors = factors, x = x)
circos.trackHist(factors = factors, x = x, col = "#999999",
border = "#999999")
circos.trackHist(factors = factors, x = x, col = "#999999",
border = "#999999")
circos.trackHist(factors = factors, x = x, bin.size = 0.1,
col = "#999999", border = "#999999")
circos.trackHist(factors = factors, x = x, draw.density = TRUE,
col = "#999999", border = "#999999")
circos.clear()

## End(Not run)
```
Usage
circos.trackLines(
  factors,
  x, y,
  track.index = get.cell.meta.data("track.index"),
  col = par("col"),
  lwd = par("lwd"),
  lty = par("lty"),
  type = "l",
  straight = FALSE,
  area = FALSE,
  area.baseline = NULL,
  border = "black",
  baseline = "bottom",
  pt.col = par("col"),
  cex = par("cex"),
  pch = par("pch"))

Arguments

  factors  A factor or a character vector which represents the categories of data
  x         Data points on x-axis
  y         Data points on y-axis
  track.index  Index for the track
  col       Line color
  lwd       line width
  lty       line style
  type      line type, similar as type argument in lines, but only in c("l","o","h","s")
  straight  whether draw straight lines between points
  area      whether to fill the area below the lines. If it is set to TRUE, col controls the filled color in the area and border controls the color of the line.
  area.baseline  deprecated, use baseline instead.
  baseline   the base line to draw area, pass to circos.lines.
  border    color for border of the area
  pt.col    if type is "o", points color
  cex       if type is "o", points size
  pch       if type is "o", points type

Details

The function adds lines in multiple cells by first splitting data into several parts in which each part corresponds to one factor (sector index) and then add lines in cells by calling circos.lines.

This function can be replaced by a for loop containing circos.lines.
Examples

# There is no example
NULL

---

**circos.trackPlotRegion**

*Create plotting regions for a whole track*

---

**Description**

Create plotting regions for a whole track

**Usage**

```r
circos.trackPlotRegion(
  factors = NULL,
  x = NULL, y = NULL,
  ylim = NULL,
  force.ylim = TRUE,
  track.index = NULL,
  track.height = circos.par("track.height"),
  track.margin = circos.par("track.margin"),
  cell.padding = circos.par("cell.padding"),
  bg.col = NA,
  bg.border = "black",
  bg.lty = par("lty"),
  bg.lwd = par("lwd"),
  panel.fun = function(x, y) {NULL})
```

**Arguments**

- `factors`: A factor or a character vector which represents categories of data, if it is NULL, then it uses all sector index.
- `x`: Data on x-axis. It is only used if `panel.fun` is set.
- `y`: Data on y-axis
- `ylim`: Range of data on y-axis
- `force.ylim`: Whether to force all cells in the track to share the same `ylim`. Normally, all cells on a same track should have same `ylim`.
- `track.index`: Index for the track which is going to be created/updated. If the specified track has already been created, this function just updated corresponding track with new plot. If the specified track is NULL or has not been created, this function just creates it. Note the value for this argument should not exceed maximum track index plus 1.
**circos.trackPlotRegion**

- **track.height**: Height of the track. It is the percentage to the radius of the unit circles. The value can be set by `uh` to an absolute unit. If updating a track (with proper `track.index` value), this argument is ignored.

- **track.margin**: only affect current track

- **cell.padding**: only affect current track

- **bg.col**: Background color for the plotting regions. It can be vector which has the same length of sectors.

- **bg.border**: Color for the border of the plotting regions. It can be vector which has the same length of sectors.

- **bg.lty**: Line style for the border of the plotting regions. It can be vector which has the same length of sectors.

- **bg.lwd**: Line width for the border of the plotting regions. It can be vector which has the same length of sectors.

- **panel.fun**: Panel function to add graphics in each cell, see "details" section and vignette for explanation.

**Details**

This function tends to be a high-level plotting function, which means, you must first call this function to create plotting regions, then those low-level graphic function such as `circos.points`, `circos.lines` can be applied.

Currently, all the cells that are created in a same track sharing same height, which means, there is no cell has larger height than others.

Since ranges for values on x-axis has already been defined by `circos.initialize`, only ranges for values on y-axis should be specified in this function. There are two ways to identify the ranges for values on y-axes either by `y` or `ylim`. If `y` is set, it must has the same length as `factors` and the `ylim` for each cell is calculated from `y` values. Also, the `ylim` can be specified from `ylim` which can be a two-element vector or a matrix which has two columns and the number of rows is the same as the length of the levels of the factors.

If there is no enough space for the new track or the new track overlaps with other tracks, there will be an error.

If `factors` does not cover all sectors, the cells in remaining unselected sectors would also be created but without drawing anything. The `ylim` for these cells are the same as that in the last created cell.

The function can also update a already-created track if the index for the track is specified. If updating an existed track, those parameters related to the position (such as track height and track margin) of the plotting region can not be changed.

**Panel**

`panel.fun` provides a convenient way to add graphics in each cell when initializing the tracks. The self-defined function needs two arguments: `x` and `y` which correspond to the data points in the current cell. When `factors`, `x`, and `y` are set in `circos.trackPlotRegion`, a subset of `x` and `y` are split by `factors` and are sent to `panel.fun` in the "current" cell. `circos.trackPlotRegion` creates plotting regions one by one on the track and `panel.fun` adds graphics in the 'current' cell after the plotting region for a certain cell has been created.

See vignette for examples of how to use this feature.
circos.trackPoints

Add points to the plotting regions in a same track

Description

Add points to the plotting regions in a same track

Usage

```r
circos.trackPoints(
  factors = NULL,
  x, y,
  track.index = get.cell.meta.data("track.index"),
  pch = par("pch"),
  col = par("col"),
  cex = par("cex"),
  bg = par("bg")
)
```

Arguments

- **factors**: A `factor` or a character vector which represents the categories of data
- **x**: Data points on x-axis
- **y**: Data points on y-axis
- **track.index**: Index for the track
- **pch**: Point type
- **col**: Point color
- **cex**: Point size
- **bg**: Background color

Examples

```r
circos.initialize(letters[1:8], xlim = c(0, 1))
set.seed(123)
df = data.frame(fa = sample(letters[1:8], 100, replace = TRUE),
                x = runif(100), y = rnorm(100))
circos.track(ylim = c(0, 1), bg.col = rand_color(8))
circos.track(df$fa, x = df$x, y = df$y, panel.fun = function(x, y) {
  circos.points(x, y)
}, track.height = 0.2, bg.border = rand_color(8))
circos.clear()
```
Details

The function adds points in multiple cells by first splitting data into several parts in which each part corresponds to one factor (sector index) and then adding points in each cell by calling `circos.points`.

Length of pch, col and cex can be one, length of levels of the factors or length of factors.

This function can be replaced by a for loop containing `circos.points`.

Examples

```r
circos.initialize(letters[1:8], xlim = c(0, 1))
df = data.frame(fa = sample(letters[1:8], 100, replace = TRUE),
                x = runif(100), y = runif(100))
circos.track(ylim = c(0, 1))
circos.trackPoints(df$fa, x = df$x, y = df$y, pch = 16, col = as.numeric(factor(df$fa)))
circos.clear()
```

`circos.trackText`  
*Draw text in cells among the whole track*

Description

Draw text in cells among the whole track

Usage

```r
circos.trackText(
    factors,
    x, y,
    labels,
    track.index = get.cell.meta.data("track.index"),
    direction = NULL,
    facing = c("inside", "outside", "reverse.clockwise", "clockwise",
              "downward", "bending", "bending.inside", "bending.outside"),
    niceFacing = FALSE,
    adj = par("adj"),
    cex = 1,
    col = par("col"),
    font = par("font")
)
```

Arguments

- **factors**: A factor or a character vector which represents the categories of data
- **x**: Data points on x-axis
- **y**: Data points on y-axis
- **labels**: Labels
- **track.index**: Index for the track
direction  deprecated, use facing instead.
facing      Facing of text
niceFacing  Should the facing of text be adjusted to fit human eyes?
adj         Adjustment for text
cex         Font size
col         Font color
font        Font style

Details

The function adds texts in multiple cells by first splitting data into several parts in which each part corresponds to one factor (sector index) and then add texts in cells by calling `circos.text`.

This function can be replaced by a `for` loop containing `circos.text`.

Examples

```r
# There is no example
NULL
```

---

circos.triangle  Draw triangles

Description

Draw triangles

Usage

```r
circos.triangle(x1, y1, x2, y2, x3, y3, ...)
```

Arguments

- `x1`  x-coordinates for the first point
- `y1`  y-coordinates for the first point
- `x2`  x-coordinates for the second point
- `y2`  y-coordinates for the second point
- `x3`  x-coordinates for the third point
- `y3`  y-coordinates for the third point
- `...` pass to `circos.polygon`
**Examples**

circos.initialize(fa = c("a", "b", "c", "d"), xlim = c(0, 10))
circos.track(ylim = c(0, 10), panel.fun = function(x, y) {
circos.triangle(c(2, 2), c(2, 8),
c(8, 8), c(2, 8),
c(5, 5), c(8, 2))
}, track.height = 0.5)

**Description**

Create plotting regions for a whole track

**Usage**

circos.update(...)  

**Arguments**

...  
pass to circos.updatePlotRegion

**Details**

shortcut function of circos.updatePlotRegion.

**Examples**

# There is no example
NULL

---

**circos.updatePlotRegion**

*Update the plotting region in an existed cell*

**Description**

Update the plotting region in an existed cell
Usage
circos.updatePlotRegion(
  sector.index = get.cell.meta.data("sector.index"),
  track.index = get.cell.meta.data("track.index"),
  bg.col = NA,
  bg.border = "black",
  bg.lty = par("lty"),
  bg.lwd = par("lwd"))

Arguments
sector.index Index for the sector
track.index Index for the track
bg.col Background color for the plotting region
bg.border Color for the border of the plotting region
bg.lty Line style for the border of the plotting region
bg.lwd Line width for the border of the plotting region

Details
You can update an existed cell by this function by erasing all the graphics. But the xlim and ylim inside the cell still remain unchanged.
Note if you use circos.track to update an already created track, you can re-define ylim in these cells.

Examples
circos.initialize(letters[1:8], xlim = c(0, 1))
circos.track(ylim = c(0, 1), panel.fun = function(x, y) {
  circos.text(CELL_META$xcenter, CELL_META$ycenter, CELL_META$sector.index)
})
circos.update(sector.index = "b", track.index = 1)
circos.rect(CELL_META$cell.xlim[1], CELL_META$cell.ylim[1],
  CELL_META$cell.xlim[2], CELL_META$cell.ylim[2],
  col = ":FF000080")
circos.clear()

circos.xaxis Draw x-axis

Description
Draw x-axis

Usage
circos.xaxis(...)

---
circos.xaxis

Draw x-axis

---

Description
Draw x-axis

Usage
circos.xaxis(...)
Arguments

all pass to `circos.axis`

Examples

```r
# There is no example
NULL
```

circos.yaxis  

*Draw y-axis*

Description

Draw y-axis

Usage

```r
circos.yaxis(
  side = c("left", "right"),
  at = NULL,
  labels = TRUE,
  tick = TRUE,
  sector.index = get.cell.meta.data("sector.index"),
  track.index = get.cell.meta.data("track.index"),
  labels.font = par("font"),
  labels.cex = par("cex"),
  labels.niceFacing = TRUE,
  tick.length = convert_x(1, "mm", sector.index, track.index),
  lwd = par("lwd"),
  col = par("col"),
  labels.col = par("col"))
```

Arguments

- **side**: add the y-axis on the left or right of the cell
- **at**: If it is numeric vector, it identifies the positions of the ticks. It can exceed ylim value and the exceeding part would be trimmed automatically.
- **labels**: labels of the ticks. The exceeding part would be trimmed automatically. The value can also be logical (either an atomic value or a vector) which represents which labels to show.
- **tick**: Whether to draw ticks.
- **sector.index**: Index for the sector
- **track.index**: Index for the track
- **labels.font**: font style for the axis labels
labels.cex: font size for the axis labels
labels.niceFacing: Should facing of axis labels be human-easy
tick.length: length of the tick
lwd: line width for ticks
col: color for the axes
labels.col: color for the labels

details
Note, you need to set the gap between sectors manually by circos.par to make sure there is enough space for y-axis.

examples

op = par(no.readonly = TRUE)
factors = letters[1:8]
circo.par(points.overflow.warning = FALSE)
circo.par(gap.degree = 8)
circo.initialize(factors = factors, xlim = c(0, 10))
circo.trackPlotRegion(factors = factors, ylim = c(0, 10), track.height = 0.5)
par(cex = 0.8)
for(a in letters[2:4]) {
circo.yaxis(side = "left", sector.index = a)
}
for(a in letters[5:7]) {
circo.yaxis(side = "right", sector.index = a)
}
circo.clear()
par(op)

col2value: Transform back from colors to values

description
Transform back from colors to values

usage

col2value(r, g, b, col_fun)
Arguments

- `r` red channel in sRGB color space, value should be between 0 and 1. The `r`, `g` and `b` arguments can be wrapped into one variable which is either a three-column matrix or a vector of colors.
- `g` green channel in sRGB color space, value should be between 0 and 1.
- `b` blue channel in sRGB color space, value should be between 0 and 1.
- `col_fun` the color mapping function generated by `colorRamp2`.

Details

colorRamp2 transforms values to colors and this function does the reversed job. Note for some color spaces, it cannot transform back to the original value perfectly.

Value

A vector of original numeric values.

Author(s)

Zuguang Gu <z.gu@dkfz.de>

Examples

```r
x = seq(0, 1, length = 11)
col_fun = colorRamp2(c(0, 0.5, 1), c("blue", "white", "red"))
col = col_fun(x)
col2value(col, col_fun = col_fun)
col2value("red", col_fun = col_fun)

col_fun = colorRamp2(c(0, 0.5, 1), c("blue", "white", "red"), space = "sRGB")
col = col_fun(x)
col2value(col, col_fun = col_fun)
```

Description

Color interpolation

Usage

```r
colorRamp2(breaks, colors, transparency = 0, space = "LAB")
```
**Arguments**

- **breaks**: A vector indicating numeric breaks
- **colors**: A vector of colors which correspond to values in breaks
- **transparency**: A single value in \([0, 1]\). 0 refers to no transparency and 1 refers to full transparency
- **space**: color space in which colors are interpolated. Value should be one of "RGB", "HSV", "HLS", "LAB", "XYZ", "sRGB", "LUV", see `color-class` for detail.

**Details**

Colors are linearly interpolated according to break values and corresponding colors through CIE Lab color space (LAB) by default. Values exceeding breaks will be assigned with corresponding maximum or minimum colors.

**Value**

It returns a function which accepts a vector of numeric values and returns interpolated colors.

**See Also**

- `col2value` converts back to the original values by providing the color mapping function generated by `colorRamp2`.

**Examples**

```r
col_fun = colorRamp2(c(-1, 0, 1), c("green", "white", "red"))
col_fun(c(-2, -1, -0.5, 0, 0.5, 1, 2))
```

---

**Description**

Convert units

**Usage**

```r
convert_height(...)  
```

**Arguments**

- `...` pass to `convert_length`

**Details**

This function is same as `convert_length`. The reason for naming this function is `convert_length` is mostly used for defining the height of tracks and track margins.
**convert_length**

**Author(s)**

Zuguang Gu <z.gu@dkfz.de>

**Examples**

```r
# see example in 'convert_length' page
NULL
```

<table>
<thead>
<tr>
<th>convert_length</th>
<th>Convert units</th>
</tr>
</thead>
</table>

**Description**

Convert units

**Usage**

```r
class_len <- convert_length(x = 1, unit = c("mm", "cm", "inches", "in"))
class_len
```

**Arguments**

- `x`: a numeric vector
- `unit`: supported units, only "mm", "cm", "inches".

**Details**

This function converts mm/cm/inches units to units measured in the canvas coordinate, e.g. how much is it in the canvas coordinate for 1 mm/cm/inches.

Since in the circular plot, the aspect ratio is always 1, it does not matter this conversion is applied on x direction or y direction.

**Author(s)**

Zuguang Gu <z.gu@dkfz.de>

**See Also**

- `convert_x` and `convert_y` convert absolute units into a data coordinate in a specified cell.

Examples

```r
fa = letters[1:10]
circos.par(cell.padding = c(0, 0, 0, 0), track.margin = c(0, 0))
circos.initialize(fa, xlim = cbind(rep(0, 10), runif(10, 0.5, 1.5)))
circos.track(ylim = c(0, 1), track.height = convert_length(5, "mm"))
circos.par(track.margin = c(0, convert_length(2, "mm")))
circos.track(ylim = c(0, 1), track.height = convert_length(1, "cm"))
circos.par(track.margin = c(0, convert_length(5, "mm")))
circos.track(ylim = c(0, 1), track.height = convert_length(1, "inches"))
circos.clear()
```

---

**convert_x**

*Convert unit on x direction in data coordinate*

**Description**

Convert unit on x direction in data coordinate

**Usage**

```r
convert_x(x, 
  unit = c("mm", "cm", "inches"),
  sector.index = get.cell.meta.data("sector.index"),
  track.index = get.cell.meta.data("track.index"),
  h = get.cell.meta.data("ycenter", sector.index = sector.index, 
    track.index = track.index))
```

**Arguments**

- `x` a numeric vector
- `unit` supported units, only "mm", "cm", "inches"
- `sector.index` index for the sector where the conversion is applied
- `track.index` index for the track where the conversion is applied
- `h` since the width of the cell is not identical from the top to the bottom in the cell, the position on y direction needs to be specified. By default it is at the middle point on y-axis

**Value**

A vector of numeric values which are measured in the specified data coordinate

**Author(s)**

Zuguang Gu <z.gu@dkfz.de>
convert_y

Convert unit on y direction in data coordinate

Description

Convert unit on y direction in data coordinate

Usage

convert_y(x,

unit = c("mm", "cm", "inches"),

sector.index = get.current.sector.index(),

track.index = get.current.track.index())

Arguments

x: a numeric vector
unit: supported units, only "mm", "cm", "inches"
sector.index: index for the sector where the conversion is applied
track.index: index for the track where the conversion is applied
Value

A vector of numeric values which are measured in the specified data coordinate

Author(s)

Zuguang Gu <z.gu@dkfz.de>

See Also

convert_x converts on x direction.


Examples

# see example on `convert_x` page
NULL

cytoband.col

Assign colors to cytogenetic band (hg19) according to the Giemsa stain results

description

Assign colors to cytogenetic band (hg19) according to the Giemsa stain results

Usage

cytoband.col(x)

Arguments

x A vector containing the Giemsa stain results

Details

The color theme is from http://circos.ca/tutorials/course/slides/session-2.pdf, page 42.

Examples

# There is no example
NULL
**degree**

*Mark the value as a degree value*

**Description**

Mark the value as a degree value

**Usage**

```r
degree(x)
```

**Arguments**

- `x` degree value

**Value**

a degree object

**Examples**

```r
# There is no example
NULL
```

---

**draw.sector**

*Draw sectors or rings in a circle*

**Description**

Draw sectors or rings in a circle

**Usage**

```r
draw.sector(  
  start.degree = 0,  
  end.degree = 360,  
  rou1 = 1,  
  rou2 = NULL,  
  center = c(0, 0),  
  clock.wise = TRUE,  
  col = NA,  
  border = "black",  
  lwd = par("lwd"),  
  lty = par("lty")
)
```
Arguments

- **start.degree**: start degree for the sector
- **end.degree**: end degree for the sector
- **rou1**: Radius for one of the arc in the sector
- **rou2**: Radius for the other arc in the sector
- **center**: Center of the circle
- **clock.wise**: The direction from start.degree to end.degree
- **col**: Filled color
- **border**: Border color
- **lwd**: Line width
- **lty**: Line style

Details

If the interval between start and end (larger or equal to 360 or smaller or equal to -360) it would draw a full circle or ring. If rou2 is set, it would draw part of a ring.

Examples

```r
generate start and end
plot(c(-1, 1), c(-1, 1), type = "n", axes = FALSE, ann = FALSE, asp = 1)
draw.sector(20, 0)
draw.sector(30, 60, rou1 = 0.8, rou2 = 0.5, clock.wise = FALSE, col = "#FF000080")
draw.sector(350, 1000, col = "#00FF0080", border = NA)
draw.sector(0, 180, rou1 = 0.25, center = c(-0.5, 0.5), border = 2, lwd = 2, lty = 2)
draw.sector(0, 360, rou1 = 0.7, rou2 = 0.6, col = "#0000FF80")

factors = letters[1:8]
circos.initialize(factors, xlim = c(0, 1))
for(i in 1:3) {
  circos.trackPlotRegion(ylim = c(0, 1))
}
circos.info(plot = TRUE)
draw.sector(get.cell.meta.data("cell.start.degree", sector.index = "a"),
get.cell.meta.data("cell.end.degree", sector.index = "a"),
rou1 = 1, col = "#FF000040")
draw.sector(0, 360,
rou1 = get.cell.meta.data("cell.top.radius", track.index = 1),
rou2 = get.cell.meta.data("cell.bottom.radius", track.index = 1),
col = "#00FF0040")
draw.sector(get.cell.meta.data("cell.start.degree", sector.index = "e"),
get.cell.meta.data("cell.end.degree", sector.index = "e"),
get.cell.meta.data("cell.top.radius", track.index = 2),
col = "#0000FF40")
```
pos = circlize(c(0.2, 0.8), c(0.2, 0.8), sector.index = "h", track.index = 2)
draw.sector(pos[1, "theta"], pos[2, "theta"], pos[1, "rou"], pos[2, "rou"],
    clock.wise = TRUE, col = "#00FFFF40")
circos.clear()

---

### fontsize

**Convert fontsize to cex**

#### Description

Convert fontsize to cex

#### Usage

```r
fontsize(x)
```

#### Arguments

- `x`
  - value for fontsize

#### Examples

```r
# There is no example
NULL
```

---

### generateRandomBed

**Generate random genomic data**

#### Description

Generate random genomic data

#### Usage

```r
generateRandomBed(
    nr = 10000,
    nc = 1,
    fun = function(k) rnorm(k, 0, 0.5),
    species = NULL)
```

#### Arguments

- `nr`
  - Number of rows
- `nc`
  - Number of numeric columns / value columns
- `fun`
  - Function for generating random values
- `species`
  - species, pass to `read.cytoband`
The function will uniformly sample positions from the genome. Chromosome names start with "chr" and positions are sorted. The final number of rows may not be exactly as same as \( r_n \).

Examples

```r
# There is no example
NULL
```

---

### genomicDensity

*Calculate genomic region density*

#### Description

Calculate genomic region density

#### Usage

```r
genomicDensity(
  region,
  window.size = 1e7,
  n.window = NULL,
  overlap = TRUE,
  count_by = c("percent", "number"),
  chr.len = NULL)
```

#### Arguments

- **region**: Genomic positions. It can be a data frame with two columns which are start positions and end positions on a single chromosome. It can also be a bed-format data frame which contains the chromosome column.
- **window.size**: Window size to calculate genomic density
- **n.window**: number of windows, if it is specified, \( window.size \) is ignored
- **overlap**: Whether two neighbouring windows have half overlap
- **count_by**: How to count the value for each window, \( \text{percent} \): percent of the window covered by the input regions; \( \text{number} \): number of regions that overlap to the window.
- **chr.len**: the chromosome length. The value should be named vector

#### Details

It calculate the percent of each genomic windows that is covered by the input regions.
Value

If the input is a two-column data frame, the function returns a data frame with three columns: start position, end position and the overlapping (value depends on the `count_by` argument). And if the input is a bed-format data frame, there will be an additionally chromosome name column.

Examples

```r
bed = generateRandomBed()
bed = subset(bed, chr == "chr1")
head(genomicDensity(bed))
head(genomicDensity(bed, count_by = "number"))
```

---

**get.all.track.index**  
*Get index for all tracks*

Description

Get index for all tracks

Usage

```r
get.all.track.index()
```

Details

It simply returns a vector of all track index.

Examples

```r
# There is no example
NULL
```
get.cell.meta.data

Get the meta data of a cell

Description

Get the meta data of a cell

Usage

get.cell.meta.data(name, sector.index = get.current.sector.index(),
                      track.index = get.current.track.index())

Arguments

- name: Only support one name at a time, see "details" section
- sector.index: Index of the sector
- track.index: Index of the track

Details

The following meta information for a cell can be obtained:

- sector.index: The name (index) for the sector
- sector.numeric.index: Numeric index for the sector
- track.index: Numeric index for the track
- xlim: Minimal and maximal values on the x-axis
- ylim: Minimal and maximal values on the y-axis
- yrange: Range of ylim
- xcenter: Center of x-axis. It equals to (xlim[2] + xlim[1])/2
- ycenter: Center of y-axis
- cell.xlim: Minimal and maximal values on the x-axis extended by cell paddings
- cell.ylim: Minimal and maximal values on the y-axis extended by cell paddings
- xplot: Degrees for right and left borders of the cell.
- yplot: Radius for top and bottom borders of the cell.
- cell.start.degree: Same as xplot[1]
- cell.end.degree: Same as xplot[2]
cell.bottom.radius  Same as yplot[1]
cell.top.radius   Same as yplot[2]
track.margin   Margin for the cell
cell.padding   Padding for the cell

The function is useful when using panel.fun in circos.track to get detailed information of the current cell.

See Also

CELL META is a short version of get.cell.meta.data.

Examples

factors = letters[1:4]
circos.initialize(factors, xlim = c(0, 1))
circos.trackPlotRegion(ylim = c(0, 1), panel.fun = function(x, y) {
  print(get.cell.meta.data("xlim"))
})
print(get.cell.meta.data("xlim", sector.index = "a", track.index = 1))
circos.clear()

get.current.chromosome

Get current chromosome name

Description

Get current chromosome name

Usage

get.current.chromosome()

Details

The function is same as get.current.sector.index and should only be put inside panel.fun when using circos.genomicTrackPlotRegion.

Examples

# There is no example
NULL
get.current.sector.index

*Get current sector index*

**Description**

Get current sector index

**Usage**

get.current.sector.index()

**Value**

Simply returns the name of current sector

**Examples**

```# There is no example
NULL```

get.current.track.index

*Get current track index*

**Description**

Get current track index

**Usage**

get.current.track.index()

**Value**

Simply returns the numeric index for the current track.

**Examples**

```# There is no example
NULL```
getI

Description
Which data that panel.fun is using

Usage
getI(...)

Arguments
...  Invisible arguments that users do not need to care

Details
The function should only be put inside panel.fun when using circos.genomicTrackPlotRegion. If stack is set to TRUE in circos.genomicTrackPlotRegion, the returned value indicates which stack the function will be applied to. If data is a list of data frames, the value indicates which data frame is being used. Please see the vignette to get a more clear explanation.

Examples
# There is no example
NULL

highlight.chromosome  Highlight chromosomes

Description
Highlight chromosomes

Usage
highlight.chromosome(...)  pass to highlight.sector
Details

This is only a shortcut function of `highlight.sector`.

Examples

```r
# There is no example
NULL
```

highlight.sector

Highlight sectors and tracks

Description

Highlight sectors and tracks

Usage

```r
highlight.sector(
  sector.index,
  track.index = get.all.track.index(),
  col = "#FF000040",
  border = NA,
  lwd = par("lwd"),
  lty = par("lty"),
  padding = c(0, 0, 0, 0),
  text = NULL,
  text.col = par("col"),
  text.vjust = 0.5,
  ...)
```

Arguments

- `sector.index`: A vector of sector index
- `track.index`: A vector of track index that you want to highlight
- `col`: Color for highlighting. Note the color should be semi-transparent.
- `border`: Border of the highlighted region
- `lwd`: Width of borders
- `lty`: Style of borders
- `padding`: Padding for the highlighted region. It should contain four values representing ratios of the width or height of the highlighted region
- `text`: text added in the highlight region, only support plotting one string at a time
- `text.vjust`: adjustment on 'vertical' (radical) direction. Besides to set it as numeric values, the value can also be a string contain absoute unit, e.g. "2.1mm", "-1 inche", but only "mm", "cm", "inches"/"inche" are allowed.
- `text.col`: color for the text
- `...`: pass to `circos.text`
You can use `circos.info` to find out index for all sectors and all tracks.

The function calls `draw.sector`.

See Also


Examples

```r
factors = letters[1:8]
circos.initialize(factors, xlim = c(0, 1))
for(i in 1:4) {
  circos.trackPlotRegion(ylim = c(0, 1))
}
circos.info(plot = TRUE)

highlight.sector(c("a", "h"), track.index = 1)
highlight.sector("c", col = "#00FF0040")
highlight.sector("d", col = NA, border = "red", lwd = 2)
highlight.sector("e", col = "#0000FF40", track.index = c(2, 3))
highlight.sector("f", "g", col = NA, border = "green",
  lwd = 2, track.index = c(2, 3))
highlight.sector(factors, col = "#FFFF0040", track.index = 4)
circos.clear()
```

---

**names.CELL_META**

Names of all meta data in the current cell

**Description**

Names of all meta data in the current cell

**Usage**

```r
## S3 method for class 'CELL_META'
names(x)
```

**Arguments**

- `x` use `CELL_META`.

**Examples**

```r
names(CELL_META)
```
posTransform.default  Genomic position transformation function

Description
Genomic position transformation function

Usage
posTransform.default(region, ...)

Arguments
region  Genomic positions at a single chromosome. It is a data frame with two columns which are start position and end position.
...
other arguments

Details
The default position transformation functions transforms position to be equally distributed along the chromosome. If users want to define their own transformation function, the requirement is that the returned value should be a data frame with two columns: transformed start position and transformed end position. The returned value should have same number of rows as the input one.

For details why need to use position transformation, please refer to circos.genomicPosTransformLines.

Examples
# There is no example
NULL

posTransform.text  Genomic position transformation function specifically for text

Description
Genomic position transformation function specifically for text
Usage

posTransform.text(
  region,
  y,
  labels,
  cex = 1,
  font = par("font"),
  sector.index = get.cell.meta.data("sector.index"),
  track.index = get.cell.meta.data("track.index"),
  padding = 0,
  extend = 0,
  ...
)

Arguments

region  Genomic positions at a single chromosome. It is a data frame with two columns which are start position and end position.
y  positions of texts
labels  text labels
cex  text size
font  text font style
sector.index  sector index
track.index  track index
padding  padding of text
extend  extend to allow labels to be put in a region which is wider than the current chromosome. The value should be a proportion value and the length is either one or two.
...
other arguments

Details

This position transformation function is designed specifically for text. Under the transformation, texts will be as close as possible to the original positions.

Examples

# There is no example
NULL
Description

Print CELL_META

Usage

## S3 method for class 'CELL_META'
print(x, ...)

Arguments

x input
...
additional parameters

Examples

# There is no example
NULL

rainfallTransform Calculate inter-distance of genomic regions

Description

Calculate inter-distance of genomic regions

Usage

rainfallTransform(
  region,
  mode = c("min", "max", "mean", "left", "right"),
  normalize_to_width = FALSE)

Arguments

region Genomic positions. It can be a data frame with two columns which are start positions and end positions on a single chromosome. It can also be a bed-format data frame which contains the chromosome column.

mode How to calculate inter-distance. For a region, there is a distance to the previous region and also there is a distance to the next region. mode controls how to merge these two distances into one value.

normalize_to_width If it is TRUE, the value is the relative distance divided by the width of the region.
**Value**

If the input is a two-column data frame, the function returns a data frame with three columns: start position, end position and distance. And if the input is a bed-format data frame, there will be the chromosome column added. The row order of the returned data frame is as same as the input one.

**Examples**

```r
bed = generateRandomBed()
bed = subset(bed, chr == "chr1")
head(rainfallTransform(bed))
```

---

**rand_color**

*Generate random colors*

**Description**

Generate random colors

**Usage**

```r
rand_color(n, hue = NULL, luminosity = "random", transparency = 0)
```

**Arguments**

- `n` number of colors
- `hue` the hue of the generated color. You can use following default color name: red, orange, yellow, green, blue, purple, pink and monochrome. If the value is a hexadecimal color string such as #00FFFF, the function will extract its hue value and use that to generate colors.
- `luminosity` controls the luminosity of the generated color. The value should be a string containing bright, light, dark and random.
- `transparency` transparency, numeric value between 0 and 1.

**Details**

The code is adapted from randomColor.js (https://github.com/davidmerfield/randomColor).

**Author(s)**

Zuguang Gu <z.gu@dkfz.de>
Examples

```r
plot(NULL, xlim = c(1, 10), ylim = c(1, 8), axes = FALSE, ann = FALSE)
points(1:10, rep(1, 10), pch = 16, cex = 5,
      col = rand_color(10, luminosity = "random"))
points(1:10, rep(2, 10), pch = 16, cex = 5,
      col = rand_color(10, luminosity = "bright"))
points(1:10, rep(3, 10), pch = 16, cex = 5,
      col = rand_color(10, luminosity = "light"))
points(1:10, rep(4, 10), pch = 16, cex = 5,
      col = rand_color(10, luminosity = "dark"))
points(1:10, rep(5, 10), pch = 16, cex = 5,
      col = rand_color(10, hue = "red", luminosity = "bright"))
points(1:10, rep(6, 10), pch = 16, cex = 5,
      col = rand_color(10, hue = "green", luminosity = "bright"))
points(1:10, rep(7, 10), pch = 16, cex = 5,
      col = rand_color(10, hue = "blue", luminosity = "bright"))
points(1:10, rep(8, 10), pch = 16, cex = 5,
      col = rand_color(10, hue = "monochrome", luminosity = "bright"))
```

read.chromInfo

Read/parse chromInfo data from a data frame/file/UCSC database

Description

Read/parse chromInfo data from a data frame/file/UCSC database

Usage

```r
read.chromInfo(
  chromInfo = system.file(package = "circlize", "extdata", "chromInfo.txt"),
  species = NULL,
  chromosome.index = usable_chromosomes(species),
  sort.chr = TRUE)
```

Arguments

- `chromInfo` Path of the chromInfo file or a data frame that already contains chromInfo data
- `species` Abbreviations of species. e.g. hg19 for human, mm10 for mouse. If this value is specified, the function will download chromInfo.txt.gz from UCSC website automatically.
- `chromosome.index` subset of chromosomes, also used to reorder chromosomes.
- `sort.chr` Whether chromosome names should be sorted (first sort by numbers then by letters). If chromosome.index is set, this argument is enforced to FALSE
Details

The function read the chromInfo data, sort the chromosome names and calculate the length of each chromosome. By default, it is human hg19 chromInfo data.

You can find the data structure for the chromInfo data from http://hgdownload.cse.ucsc.edu/goldenpath/hg19/database/chromInfo.txt.gz

Value

df Data frame for chromInfo data (rows are sorted if sort.chr is set to TRUE)
chromosome Sorted chromosome names
chr.len Length of chromosomes. Order are same as chromosome

Examples

data = read.chromInfo(species = "hg19")
data = read.chromInfo(chromInfo = system.file(package = "circlize", "extdata", "chromInfo.txt"))
chromInfo = read.table(system.file(package = "circlize", "extdata", "chromInfo.txt"),
colClasses = c("character", "numeric"), sep = "\t")
data = read.chromInfo(chromInfo = chromInfo)

Description

Read/parse cytoband data from a data frame/file/UCSC database

Usage

read.cytoband(
cytoband = system.file(package = "circlize", "extdata", "cytoBand.txt"),
species = NULL,
chromosome.index = usable_chromosomes(species),
sort.chr = TRUE)

Arguments

cytoband Path of the cytoband file or a data frame that already contains cytoband data
species Abbreviations of species. e.g. hg19 for human, mm10 for mouse. If this value is specified, the function will download cytoBand.txt.gz from UCSC website automatically.
chromosome.index subset of chromosomes, also used to reorder chromosomes.
sort.chr Whether chromosome names should be sorted (first sort by numbers then by letters). If chromosome.index is set, this argument is enforced to FALSE
reverse.circlize

Details

The function read the cytoband data, sort the chromosome names and calculate the length of each chromosome. By default, it is human hg19 cytoband data.

You can find the data structure of the cytoband data from http://hgdownload.cse.ucsc.edu/goldenpath/hg19/database/cytoBand.txt.gz

Value

df  Data frame for cytoband data (rows are sorted if sort.chr is set to TRUE)
chromosome  Sorted chromosome names
chr.len  Length of chromosomes. Orders are same as chromosome

Examples

data = read.cytoband(species = "hg19")
data = read.cytoband(cytoband = system.file(package = "circlize", "extdata", "cytoBand.txt"))
cytoband = read.table(system.file(package = "circlize", "extdata", "cytoBand.txt"),
colClasses = c("character", "numeric", "numeric", "character", "character"), sep = "\t")
data = read.cytoband(cytoband = cytoband)

reverse.circlize

Convert to data coordinate system

Description

Convert to data coordinate system

Usage

reverse.circlize(
x, y,
sector.index = get.current.sector.index(),
track.index = get.current.track.index())

Arguments

x  degree values. The value can also be a two-column matrix/data frame if you put x and y data points into one variable.
y  distance to the circle center (the radius)
sector.index  Index for the sector where the data coordinate is used
track.index  Index for the track where the data coordinate is used

Details

This is the reverse function of circlize. It transform data points from polar coordinate system to a specified data coordinate system.
Value

A matrix with two columns (x and y)

Examples

```r
pdf(NULL)
factors = letters[1:4]
circos.initialize(factors, xlim = c(0, 1))
circos.trackPlotRegion(ylim = c(0, 1))
reverse.circlize(c(30, 60), c(0.9, 0.8))
reverse.circlize(c(30, 60), c(0.9, 0.8), sector.index = "d", track.index = 1)
reverse.circlize(c(30, 60), c(0.9, 0.8), sector.index = "a", track.index = 1)
circos.clear()
dev.off()
```

Description

Set flag to current cell

Usage

```r
set.current.cell(sector.index, track.index)
```

Arguments

- `sector.index`: sector index
- `track.index`: track index

Details

After setting the current cell, all functions which need `sector.index` and `track.index` arguments and are applied to the current cell do not need to specify the two arguments explicitly.

Examples

```r
pdf(NULL)
circos.initialize(letters[1:8], xlim = c(0, 1))
circos.track(ylim = c(0, 1))
circos.info()
set.current.cell("b", 1)
circos.info()
circos.clear()
dev.off()
```
show.index

Label the sector index and the track index on each cell

Description

Label the sector index and the track index on each cell

Usage

show.index()

Details

This function is deprecated, please use `circos.info` instead.

Examples

```r
# There is no example
NULL
```

smartAlign

Adjust positions of text

Description

Adjust positions of text

Usage

smartAlign(x1, x2, xlim)

Arguments

- `x1` position which corresponds to the top of the text
- `x2` position which corresponds to the bottom of the text
- `xlim` ranges on x-axis

Details

used internally

Examples

```r
# There is no example
NULL
```
uh  

Convert units

Description
Convert units

Usage
uh(...)

Arguments
... pass to convert_length

Details
This function is same as convert_length.

Author(s)
Zuguang Gu <z.gu@dkfz.de>

Examples

# see example in `convert_length` page
NULL

ux  

Convert unit on x direction in data coordinate

Description
Convert unit on x direction in data coordinate

Usage
ux(...)

Arguments
... pass to convert_x

Details
This function is same as convert_x.
Author(s)

Zuguang Gu <z.gu@dkfz.de>

Examples

# see example in `convert_x` page
NULL

#### uy

Convert unit on y direction in data coordinate

Description

Convert unit on y direction in data coordinate

Usage

uy(...)

Arguments

... pass to convert_y

Details

This function is same as convert_y.

Author(s)

Zuguang Gu <z.gu@dkfz.de>

Examples

# see example in `convert_y` page
NULL
$.CELL_META

Easy to way to get meta data in the current cell

Description

Easy to way to get meta data in the current cell

Usage

```
## S3 method for class 'CELL_META'

x$name
```

Arguments

- `x` name of the variable should be "CELL_META"
- `name` name of the cell meta name

Details

The variable `CELL_META` can only be used to get meta data of the "current" cell. Basically you can simply replace e.g. `get.cell.meta.data("sector.index")` to `CELL_META$sector.index`.

See Also

- `get.cell.meta.data`

Examples

```r
# There is no example
NULL
```
Index

$.CELL_MEGA, 109
add_transparency, 6
adjacencyList2Matrix, 6
Arrowhead, 14, 17, 57
as.raster, 64
calc_gap, 7
CELL_META, 8, 8, 93, 97, 109
chordDiagram, 5, 8, 15, 18
chordDiagramFromDataFrame, 9, 10, 12, 18
chordDiagramFromMatrix, 9, 10, 15
circlize, 19, 104
circlize-package, 4
circos.arrow, 20
circos.axis, 4, 21, 27, 79
circos.clear, 4, 24, 59
circos.dendrogram, 25
circos.genomicAxis, 26
circos.genomicDensity, 5, 27
circos.genomicHeatmap, 5, 29
circos.genomicIdeogram, 5, 30
circos.genomicInitialize, 5, 31, 52, 53
circos.genomicLabels, 5, 33
circos.genomicLines, 5, 34
circos.genomicLink, 5, 37
circos.genomicPoints, 5, 38
circos.genomicPostTransformLines, 40, 98
circos.genomicRainfall, 5, 41
circos.genomicRect, 5, 43
circos.genomicText, 5, 45
circos.genomicTrack, 5, 47
circos.genomicTrackPlotRegion, 36, 39, 44, 47, 48, 93, 95
circos.info, 4, 50, 97, 106
circos.initialize, 4, 25, 32, 51, 61, 73
circos.initializeWithIdeogram, 5, 52
circos.lines, 4, 28, 35, 36, 54, 71, 73
circos.link, 4, 10, 14, 17, 18, 37, 56
circos.nested, 58, 59
circos.par, 4, 59, 80
circos.plot, 4, 38, 39, 61, 62, 73, 75
circos.polygon, 4, 62, 76
circos.raster, 63
circos.rect, 4, 43, 46, 65, 65
circos.segments, 4, 66
circos.text, 4, 22, 46, 67, 76, 96
circos.track, 4, 68, 78, 93
circos.trackHist, 69
circos.trackLines, 4, 70
circos.trackPlotRegion, 28, 41, 42, 48, 49, 62, 68, 72, 73
circos.trackPoints, 4, 74
circos.trackText, 4, 75
circos.triangle, 76
circos.update, 4, 77
circos.updatePlotRegion, 77, 77
circos.xaxis, 4, 78
circos.yaxis, 4, 23, 79
col2value, 80, 82
colorRamp2, 13, 16, 29, 81, 81, 82
convert_height, 58, 60, 82
convert_length, 82, 83, 107
convert_x, 83, 84, 86, 107
convert_y, 23, 83, 85, 85, 108
cytoBand.col, 86
degree, 67, 87
dendrogram, 25
draw.sector, 87, 97
factor, 51, 71, 72, 74, 75
fontsize, 89
generateRandomBed, 89
genomicDensity, 28, 90
get.all.chromosome, 91
get.all.track.index, 91
generate.cellMeta.data, 8, 92, 93, 109
generate.current.chromosome, 93
get.current.sector.index, 93, 94
get.current.track.index, 94
gti, 49, 95

highlight.chromosome, 95
highlight.sector, 95, 96, 96
hist, 70

LAB, 82
lines, 55, 66, 71

names.CELL_META, 97
par, 60, 62
plot, 51
points, 62
polygon, 20, 62, 63, 65
posTransform.default, 35, 39, 40, 43, 46, 98
posTransform.text, 46, 98
print.CELL_META, 100

rainfallTransform, 42, 100
rand_color, 101
read.chromInfo, 52, 102
read.cytoband, 31, 52, 53, 89, 103
reverse.circlize, 104

set.current.cell, 105
show.index, 106
smartAlign, 106
sRGB, 81
Subset.CELL_META ($.CELL_META), 109

text, 67

uh, 58, 73, 107
ux, 107
uy, 108