Package ‘circlize’

December 22, 2015

Type Package
Title Circular Visualization
Version 0.3.4
Date 2015-12-21
Author Zuguang Gu
Maintainer Zuguang Gu <z.gu@dkfz.de>
Depends R (>= 2.10.0), graphics
Imports GlobalOptions (>= 0.0.6), shape, grDevices, utils, stats, colorspace, methods, grid
Suggests knitr, dendextend (>= 1.0.1)
VignetteBuilder knitr

Description Circular layout is an efficient way for the visualization of huge amounts of information. Here the circlize package provides an implementation of circular layout generation in R as well as an enhancement of available software. The flexibility of this 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, circlize gives users more convenience and freedom to design figures for better understanding complex patterns behind multi-dimensional data.

URL https://github.com/jokergoo/circlize
License GPL (>= 2)
Repository CRAN
Date/Publication 2015-12-22 14:36:53

R topics documented:
circlize-package ................................................................. 3
adjacencyList2Matrix ......................................................... 5
R topics documented:

chordDiagram .......................................................... 5
chordDiagramFromDataFrame ..................................... 7
chordDiagramFromMatrix ........................................ 10
circlize ................................................................. 12
circos.axis ............................................................. 13
circos.clear ............................................................. 15
circos.dendrogram ................................................ 16
circos.genomicDensity ............................................ 17
circos.genomicInitialize ......................................... 19
circos.genomicLines ............................................... 20
circos.genomicLink ............................................... 22
circos.genomicPoints ............................................. 24
circos.genomicPosTransformLines ............................. 26
circos.genomicRainfall ........................................... 28
circos.genomicRect ................................................. 29
circos.genomicText ................................................ 31
circos.genomicTrack .............................................. 33
circos.genomicTrackPlotRegion ................................. 34
circos.info ............................................................. 36
circos.initialize .................................................... 37
circos.initializeWithIdeogram ................................ 38
circos.lines ........................................................... 40
circos.link ............................................................. 42
circos.par ............................................................. 44
circos.points ........................................................ 46
circos.polygon ....................................................... 47
circos.rect ............................................................ 48
circos.segments ..................................................... 49
circos.text ............................................................ 49
circos.track .......................................................... 51
circos.trackHist ..................................................... 52
circos.trackLines .................................................. 53
circos.trackPlotRegion ............................................ 54
circos.trackPoints ................................................ 56
circos.trackText .................................................... 57
circos.update ........................................................ 58
circos.updatePlotRegion .......................................... 59
circos.xaxis ........................................................... 60
circos.yaxis ........................................................... 60
col2value .............................................................. 61
colorRamp2 ............................................................ 62
cytoband.col ........................................................ 63
draw.sector ............................................................ 64
generateRandomBed ................................................. 65
genomicDensity ....................................................... 66
get.all.sector.index ............................................... 67
get.all.track.index ................................................ 68
get.cell.meta.data .................................................. 69
Description

Circular layout 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 graphical functions:

- `circos.points`
- `circos.lines`
- `circos.rect`
- `circos.polygon`
- `circos.text`
- `circos.axis`
- `circos.link`, This maybe the unique feature for circular layout to represent relationships between elements.

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.trackPlotRegion`
- `circos.updatePlotRegion`
- `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`
- `circos.genomicInitialize`

Functions to arrange genomic circular layout:

- `circos.genomicTrackPlotRegion`

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`

Finally, function that draws chord diagram:

- `chordDiagram`
- `chordDiagramFromMatrix`
- `chordDiagramFromDataFrame`

Please refer to the vignettes to find out how to draw basic and advanced circular plots by this package.

**Examples**

```r
# There is no example
NULL
```
adjacencyList2Matrix

Convert adjacency list to adjacency matrix

Description

Convert adjacency list to adjacency matrix

Usage

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

lt = data.frame(letters[1:5], letters[6:10])
adjacencyList2Matrix(lt)

lt = data.frame(letters[1:5], letters[6:10], 1:5)
adjacencyList2Matrix(lt)

set.seed(123)
lt = data.frame(sample(letters, 4), sample(letters, 4), 1:4)
adjacencyList2Matrix(lt)
adjacencyList2Matrix(lt, square = TRUE)

chordDiagram

Plot Chord Diagram

Description

Plot Chord Diagram

Usage

chordDiagram(x, ...)
Arguments

x  a matrix or a data frame. The function will pass all argument to `chordDiagramFromMatrix` or `chordDiagramFromDataFrame` depending on the type of x

...  pass to `chordDiagramFromMatrix` or `chordDiagramFromDataFrame`.

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.

References


Examples

```r
## Not run:

########################################################################## example 1 ##############################################################################
set.seed(123)
mat = matrix(sample(1:100, 18, replace = TRUE), 3, 6)
rownames(mat) = letters[1:3]
colnames(mat) = LETTERS[1:6]

### basic settings
par(mfrow = c(3, 2))
par(mar = c(1, 1, 1, 1))

chordDiagram(mat)
circos.clear()

circos.par(gap.degree = c(rep(2, nrow(mat)-1), 10, rep(2, ncol(mat)-1), 10))
chordDiagram(mat)
circos.clear()

circos.par(start.degree = 90)
chordDiagram(mat)
circos.clear()

chordDiagram(mat, order = c("A", "B", "a", "C", "D", "b", "E", "F", "c"))
chordDiagram(mat, directional = TRUE)
chordDiagram(mat, directional = TRUE, diffHeight = 0.06)
circos.clear()

########################################################################## example 2 ##########################################################################
set.seed(123)
```
chordDiagramFromDataFrame

mat = matrix(sample(1:100, 18, replace = TRUE), 3, 6)
rownames(mat) = letters[1:3]
colnames(mat) = LETTERS[1:6]

### colors settings
rand_color = function(n, alpha = 1) {
  return(rgb(runif(n), runif(n), runif(n), alpha = alpha))
}
par(mfrow = c(3, 3))
par(mar = c(1, 1, 1, 1))
grid.col = NULL
grid.col[letters[1:3]] = c("red", "green", "blue")
grid.col[LETTERS[1:6]] = "grey"
chordDiagram(mat, grid.col = grid.col)
chordDiagram(mat, grid.col = grid.col, transparency = 0.5)
col_mat = rand_color(length(mat), alpha = 0.5)
dim(col_mat) = dim(mat)
chordDiagram(mat, grid.col = grid.col, col = col_mat)
chordDiagram(mat, grid.col = grid.col, col = colorRamp2(quantile(mat, seq(0, 1, by = 0.1)), rev(heat.colors(11))), transparency = 0.5)

chordDiagram(mat, grid.col = grid.col, row.col = 1:3, transparency = 0.5)
chordDiagram(mat, grid.col = grid.col, column.col = 1:6, transparency = 0.5)
chordDiagram(mat, grid.col = grid.col, row.col = c("FF000080", "00FF0010", "0000FF10"))
circos.clear()

## End(Not run)

circularDiagramFromDataFrame

Plot Chord Diagram from a data frame

Description

Plot Chord Diagram from a data frame

Usage

chordDiagramFromDataFrame(df, grid.col = NULL, grid.border = NA, transparency = 0.5, col = NULL, order = NULL, directional = 0, direction.type = "diffHeight", diffHeight = 0.04, reduce = 1e-5, self.link = 2, preAllocateTracks = NULL, annotationTrack = c("name", "grid", "axis"), annotationTrackHeight = c(0.05, 0.05), 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"), ...)

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.

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.

`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`

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

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

`link.sort`  
whether sort links on every sector based on the width of the links on it.

`link.decreasing`  
for `link.sort`

`link.arr.length`  
pass to `circos.link`, same settings as `link.lwd`.

`link.arr.width`  
pass to `Arrowhead`, same settings 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, same settings as `link.lwd`.

`link.arr.lwd`  
line width of the single line link which is put in the center of the belt, same settings as `link.lwd`.

`link.arr.lty`  
line type of the single line link which is put in the center of the belt, same settings as `link.lwd`.

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

**Details**

...

**Examples**

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

Plot Chord Diagram from a matrix

Description

Plot Chord Diagram from a matrix

Usage

chordDiagramFromMatrix(mat, grid.col = NULL, grid.border = NA, transparency = 0.5, 
col = NULL, row.col = NULL, column.col = NULL, order = NULL, directional = 0, 
direction.type = "diffHeight", diffHeight = 0.04, reduce = 1e-5, self.link = 2, 
symmetric = FALSE, keep.diagonal = FALSE, preAllocateTracks = NULL, 
annotationTrack = c("name", "grid", "axis"), annotationTrackHeight = c(0.05, 0.05), 
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"), ...)

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 trans-
parency. 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`.
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.
link.decreasing  for `link.sort`.
link.arr.length  pass to `circos.link`, same settings as `link.lwd`. 
link.arr.width: pass to Arrowhead, same settings 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, same settings as link.lwd.
link.arr.lwd: line width of the single line link which is put in the center of the belt, same settings as link.lwd.
link.arr.lty: line type of the single line link which is put in the center of the belt, same settings as link.lwd.
... pass to circos.link

Details

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

Examples

# There is no example
NULL

---
circlize

Return the coordinate in polar coordinate system

Description

Return the coordinate in 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
y Data points on y-axis
sector.index Index for the sector
track.index Index for the track

Details

This is the core function in the package. It transform data points from data coordinate system to polar coordinate system.
Value

A matrix with two columns (theta and rou)

Examples

# There is no example
NULL

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"),
layers.font = par("font"), layers.cex = par("cex"),
layers.facing = "inside", layers.direction = NULL, layers.niceFacing = TRUE,
direction = c("outside", "inside"), minor.ticks = 4,
major.tick.percentage = 0.1, layers.away.percentage = major.tick.percentage/2,
lwd = par("lwd")

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.
layers   labels of the major ticks. Also, the exceeding part would be trimmed automati-
          cally.
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
layers.font  font style for the axis labels
layers.cex   font size for the axis labels
layers.direction
              deprecated, use facing instead.
layers.facing facing of labels on axis, passing to circos.text
layers.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 Length of the major ticks. It is the percentage to the height of the cell.
labels.away.percentage The distance for the axis labels to the major ticks. It is the percentage to the height of the cell.
lwd line width for ticks

details It can only draw axes on x-direction.


examples

```r
## Not run:
library(circlize)

par(mar = c(1, 1, 1, 1))
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()
```

--------------------------------------------------------------------------------

real-time clock
factors = letters[1]
par(mar = c(1, 1, 1, 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
    drawsector(roul = 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()

## End(Not run)

---

**circos.clear**

*Reset the circos layout parameters*

**Description**

Reset the circos layout parameters

**Usage**

```r
circos.clear()
```
Details

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

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

References


Examples

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

**circos.dendrogram** Add circlized dendrograms

Description

Add circlized dendrograms

Usage

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

Arguments

- `dend` A `dendrogram` object.
- `facing` Is the dendromgrams 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.

Details

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

You can use the `dendextend` package to render the dendrograms.
Examples

```r
# Not run:
load(paste0(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 = c[t[labels[i]]], cex = 0.7)
    }
  })

require(dendextend)
dend = color_branches(dend, k = 6, col = 1:6)
circos.trackPlotRegion(ylim = c(0, max_height), bg.border = NA,
  track.height = 0.3, panel.fun = function(x, y) {
    circos.dendrogram(dend, max_height = max_height)
  })
circos.clear()

# End(Not run)
```

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,
  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
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
...

Details

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

References


Examples

```r
## Not run:
library(circlize)

par(mar = c(1, 1, 1, 1))

load(paste(system.file(package = "circlize"), 
"/extdata/DMR.RData", sep=""))

# 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.genomicInitialize

**Initialize circos plot with any genomic data**

**Description**

Initialize circos plot with any genomic data

**Usage**

```r
circos.genomicInitialize(data, sector.names = NULL, major.by = NULL,
plotType = c("axis", "labels"), tickLabelsStartFromZero = TRUE,
track.height = 0.05, ...)
```

**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.
- **track.height**: If PlotType is not NULL, height of the annotation track.
- ... Pass to `circos.initialize`

**Details**

The function will initialize circos 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.

**References**

Examples

```r
## Not run:

df = read.cytoband()

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()

## End(Not run)
```

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
postTransform Self-defined function to transform genomic positions, see postTransform.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
pt.col        Settings are similar as col. Pass to circos.lines
cex           Settings are similar as col. Pass to circos.lines
pch           Settings are similar as col. Pass to circos.lines
...           mysterious parameters

Details

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

References


Examples

## Not run:

par(mar = c(1, 1, 1, 1))

## test line
```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()

## End(Not run)
```

circos.genomicLink 

Add links from two sets of genomic positions

Description

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 = NA, ...)

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
...             Pass to circos.link

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.

References


Examples

## Not run:
set.seed(123)

library(circlize)

par(mar = c(1, 1, 1, 1))
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)
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"), ...)
```

**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`
- `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`
- `...`: Mysterious parameters

**Details**

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

**References**

Examples

```r
## Not run:
par(mar = c(1, 1, 1, 1))

circos.par("track.height" = 0.1)
circos.initializeWithIdiogram(plotType = NULL)

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

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

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

# data frame list
circos.genomicTrackPlotRegion(bet_list, 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, ...)
})

circos.genomicTrackPlotRegion(bet_list, 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")
})

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.genomicPosTransformLines

Add genomic position transformation lines between tracks

Description
Add genomic position transformation lines between tracks

Usage
com.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.
 References


 Examples

 ```
 ## Not run:
 library(circlize)

 par(mfrow = c(2, 1))
 par(mar = c(1, 1, 1, 1))
 ### rect matrix
 circos.par(cell.padding = c(0, 0, 0, 0))
 circos.initializeWithIdeogram()

 bed = generateRandomBed(nr = 100, nc = 4)
 circos.genomicPosTransformLines(bed, posTransform = posTransform.default,
  horizontalline = "top", track.height = 0.1)

 f = colorRamp2(breaks = c(-1, 0, 1), colors = c("green", "black", "red"))
 circos.genomicTrackPlotRegion(bed, stack = TRUE, panel.fun = function(region, value, ...) {
  circos.genomicRect(region, value, col = f(value[[1]]),
    border = f(value[[1]]), posTransform = posTransform.default, ...)
}, bg.border = NA)

 circos.clear()

 circos.par(cell.padding = c(0, 0, 0, 0))
 circos.initializeWithIdeogram(plotType = NULL)

 bed = generateRandomBed(nr = 20, nc = 4)
 circos.genomicTrackPlotRegion(bed, ylim = c(0, 1), panel.fun = function(region, value, ...) {
  circos.genomicText(region, value, y = 0, adj = c(1, 0.5),
    labels = "gene", facing = "reverse.clockwise",
    posTransform = posTransform.default)
}, bg.border = NA)

 circos.genomicPosTransformLines(bed, posTransform = posTransform.default,
  horizontalline = "bottom", direction = "outside", track.height = 0.1)

 cytoband = read.cytoband()$df
 circos.genomicTrackPlotRegion(cytoband, stack = TRUE, panel.fun = function(region, value, ...) {
  circos.genomicRect(region, value, col = cytoband.col(value$V5), border = NA, ...)
}, track.height = 0.05)

 circos.clear()

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

Genomic rainfall plot

Description
Genomic rainfall plot

Usage
circos.genomicRainfall(data, ylim = c(0, 9), col = "black", pch = par("pch"),
cex = par("cex"), ...)

Arguments
data A bed-file-like data frame or a list of data frames
ylim ylim for rainfall plot track. It’s value should be log10(inter-distance+1)
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
... Pass to circos.trackPlotRegion

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.

References

Examples
## Not run:
library(circlize)
par(mar = c(1, 1, 1, 1))
load(paste(system.file(package = "circlize"), "/extdata/DMR.RData", sep=""))
# rainfall
circos.initializeWithIdeogram(plotType = c("axis", "labels"))
```r
bed_list = list(DMR_hyper, DMR_hypo)
circos.genomicRainfall(bed_list, pch = 16, cex = 0.4, col = c("FF000000", "0000FF80"))
circos.genomicDensity(bed_list[[1]], col = c("FF000080"), track.height = 0.1)
circos.genomicDensity(bed_list[[2]], col = c("0000FF08"), 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**

```r
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"), lwd = par("lwd"), ...)
```

**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`
- `lwd` 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 `circos.genomicTrackPlotRegion`.

References


Examples

```R
## Not run:

circlize implements and enhances circular visualization in R.

```
circos.genomicText

```
circos.initializeWithIdeogram(plotType = NULL)
bed1 = generateRandomBed(nr = 100)
bed2 = generateRandomBed(nr = 100)
bed_list = list(bed1, bed2)
f = colorRamp2(breaks = c(-1, 0, 1), colors = c("green", "black", "red"))
circos.genomicTrackPlotRegion(bed_list, stack = TRUE,
panel.fun = function(region, value, ...) {
  circos.genomicRect(region, value, col = f(value[[1]]),
                     border = NA, ...)
  i = getI(...)
  cell.xlim = get.cell.meta.data("cell.xlim")
  circos.lines(cell.xlim, c(i, i), lty = 2, col = "#000000")
})
circos.genomicTrackPlotRegion(bed_list, ylim = c(0, 3),
panel.fun = function(region, value, ...) {
  i = getI(...)
  circos.genomicRect(region, value, ytop = i+0.4, ybottom = i-0.4, col = f(value[[1]]),
                     border = NA, ...)
  cell.xlim = get.cell.meta.data("cell.xlim")
  circos.lines(cell.xlim, c(i, i), lty = 2, col = "#000000")
})
circos.genomicTrackPlotRegion(bed1, panel.fun = function(region, value, ...) {
  circos.genomicRect(region, value, col = "red", border = NA, ...)
})
circos.genomicTrackPlotRegion(bed_list, panel.fun = function(region, value, ...) {
  i = getI(...)
  circos.genomicRect(region, value, col = i, border = NA, ...)
})
circos.clear()

## End(Not run)
```

---

circos.genomicText  \hspace{1cm} \textit{Draw text in a cell, specifically for genomic graphics}

\textbf{Description}

Draw text in a cell, specifically for genomic graphics
Usage

```r
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, ...)```

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`
- `...` Mysterious parameters

Details

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

References

### Examples

```r
# Not run:
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()

# End(Not run)
```

---

**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
```
**circos.genomicTrackPlotRegion**

*Create a track for genomic graphics*

**Description**

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 not 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 splitted 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`.

References


Examples

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

Get information of the circos plot

Usage

```r
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 for 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.

References


Examples

```r
## Not run:
library(circlize)
factors = letters[1:4]
circos.initialize(factors, xlim = c(0, 1))
circos.trackPlotRegion(ylim = c(0, 1))
circos.trackPlotRegion(ylim = c(0, 1))
circos.info(sector.index = "a", track.index = 1)
circos.info(sector.index = "a", track.index = 1:2)
circos.info(sector.index = c("a", "b"), track.index = 1)
circos.info(sector.index = "a")
circos.info()
circos.info(plot = TRUE)
circos.clear()
```
**circos.initialize**   \[ Initialize the circos layout \]

**Description**

Initialize the circos layout

**Usage**

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

**Arguments**

- `factors`: Factors which represent data categories
- `x`: Data on x-axis, a vector
- `xlim`: Limitations for values on x-axis
- `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 which is calculated internally.

**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.

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` and `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 (number of levels) 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.

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. Anyway, in some cases, it is useful to manually set the width such as you want to zoom in some part of the sectors.

The function finally calls `plot` and be ready for adding graphics.
References


Examples

```r
## Not run:
circos.initialize(factors = sample(letters[1:4], 20, replace = TRUE), xlim = c(0, 1))
circos.clear()

circos.initialize(factors = sample(letters[1:4], 20, replace = TRUE), xlim = cbind(1:4, 1:4*2))
circos.clear()

circos.initialize(factors = sample(letters[1:4], 20, replace = TRUE), x = rnorm(20))
circos.clear()

## End(Not run)
```

circos.initializeWithIdeogram

*Initialize the circos layout with an ideogram*

Description

Initialize the circos layout with an ideogram

Usage

```r
circos.initializeWithIdeogram(cytoband = paste(system.file(package = "circlize"),
                          "/extdata/cytoBand.txt", sep=""), species = NULL, sort.chr = TRUE,
chromosome.index = NULL, major.by = NULL,
plotType = c("ideogram", "axis", "labels"),
track.height = 0.05, ideogram.height = 0.05, ...)
```

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` and `read.chromInfo`.
- **chromosome.index** subset of chromosomes, also used to re-set chromosome orders.
- **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.

**track.height**

Height of the track which contains "axis" and "labels".

**ideogram.height**

Height of the ideogram track

**...**

Pass to `circos.initialize`

**Details**

The function will initialize the circos 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 circos plot.

The style of ideogram is almost fixed, but you can customize it with your self-defined code. Refer to vignette for demonstration.

**References**


**Examples**

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

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

cyto.band.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"))

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

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

cytoband = 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**

```r
circos.lines(x, y, sector.index = get.cell.meta.data("sector.index"),
             track.index = get.cell.meta.data("track.index"),
             col = ifelse(area, "grey", "black"),
             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

- `y`  
  Data points on y-axis

- `sector.index`  
  Index for the sector

- `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 color of the line.
area.baseline: Deprecated, use baseline instead.

baseline: 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 bottom and top. This argument also works if type is set to h.

border: Color for border of the area

pt.col: If type is "o", point color

cex: If type is "o", point size

pch: If type is "o", point type

Details

Normally, straight lines in the Cartesian coordinate have to be transformed into curves in the circos 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.

Draw 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.

References


Examples

```r
# Not run:
library(circlize)
par(mar = c(1, 1, 1, 1), cex = 0.6)
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 = 'f'", 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\nbaseline = 'top'", sector.index = "i")

circos.clear()
par(cex = 1)

## End(Not run)

circos.link  
\textit{Draw links between points or intervals}

\textbf{Description}

Draw links between points or intervals

\textbf{Usage}

\begin{verbatim}
circos.link(sector.index1, point1, sector.index2, point2,  
  rou = get_most_inside_radius(),  
  rou1 = rou, rou2 = rou, h = NULL, w1 = 1, h2 = h, w2 = w,  
  col = "black", lwd = par("lwd"), lty = par("lty"), border = NA,  
  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)
\end{verbatim}

\textbf{Arguments}

- \texttt{sector.index1}  
  Index for the first sector
- \texttt{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.
- \texttt{sector.index2}  
  Index for the other sector
- \texttt{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.
- \texttt{rou}  
  The position of the 'root' of the link. 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.
- \texttt{rou1}  
  The position of root 1 of the link.
- \texttt{rou2}  
  The position of root 2 of the link.
- \texttt{h}  
  Height of the link.
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
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.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 that is not passed to Arrowhead (big.arrow).
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.

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.

See vignette for detailed explanation.

References


Examples

## Not run:

par(mar = c(1, 1, 1, 1))
factors = letters[1:8]
circos.initialize(factors = factors, xlim = c(0, 10))
circos.trackPlotRegion(factors = factors, ylim = c(0, 1), bg.col = "grey", bg.border = NA, track.height = 0.05)
circos.info(plot = TRUE)
circos.link("a", 5, "c", 5, rou1 = 0.4, rou2 = 0.6, col = "black")
circos.link("a", 5, "g", 5, col = "black", h = 0.5, w = -0.25)
circos.link("c", 10, "d", c(1, 4), col = "#00000040", border = "black")
circos.link("a", c(2, 8), "g", c(4, 4.5), rou1 = 0.9, rou2 = 0.8, col = "#00000040", border = "black")
```r
circos.link("b", c(1, 10), "a", c(1, 10), rou1 = 0.9, rou2 = 0.4,
col = "#00000040", border = "black")
circos.clear()

par(mar = c(1, 1, 1, 1))
factors = letters[1:8]
circos.par("canvas.xlim" = c(-2, 2), "canvas.ylim" = c(-2, 2))
circos.initialize(factors = factors, xlim = c(0, 10))
circos.trackPlotRegion(factors = factors, ylim = c(0, 1), bg.col = "grey",
  bg.border = NA, track.height = 0.05)
circos.info(plot = TRUE)
circos.link("a", 5, "b", 5, col = "black", w = 1)
circos.link("b", 5, "c", 5, col = "black", w = 2)
circos.link("c", 5, "d", 5, col = "black", w = 0.25)
circos.link("d", 5, "e", 5, col = "black", w = -0.25)
circos.clear()

## End(Not run)
```

## Parameters for circos layout

### Description

Parameters for circos layout

### Usage

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

### Arguments

- `...`: Arguments for the parameters, see "details" section
- `RESET`: reset to default values
- `READ.ONLY`: whether only return read-only options

### Details

Global parameters for the circos 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.
**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.

**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 (2πi) 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.

**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.

**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 but print 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 circos layout (i.e. before calling circos.initialize) because these values will not be changed after adding sectors on the circle. The left and right padding for cell.padding will also be ignored after the initialization because all cells in a sector would share the same left and right paddings.

References


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"))
```

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
- `pch`: Point type
- `col`: Point color
- `cex`: Point size

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 as the normal `points` function, just adding points in the plotting region. The position of plotting region 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`.

References


Examples

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

References


Examples

```r
## Not run:
library(circlize)
set.seed(123)
par(mar = c(1, 1, 1, 1))
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,
```
```r
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.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, ...)

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
- ...: pass to lines

Examples
# There is no example
NULL

circos.text

Draw text in a cell

Description
Draw text in a cell

Usage
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 = "black", 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  Adjustment for text
cex  Font size
col  Font color
font  Font style
...  Pass to text

Details

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

References


Examples

```r
## Not run:
library(circlize)
par(mar = c(1, 1, 1, 1), mfrow = c(2, 1))
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, 9, "inside", facing = "inside", cex = 0.8)
circos.text(7, 9, "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(3, 9, "==bending.inside==", facing = "bending.inside", cex = 0.8)
circos.text(7, 9, "==bending.outside==", facing = "bending.outside", cex = 0.8)
})
circos.clear()

factors = LETTERS[1:20]
circos.par(points.overflow.warning = FALSE)
circos.initialize(factors = factors, xlim = c(0, 1))
circos.trackPlotRegion(factors = factors, ylim = c(0, 1), track.height = 0.5,
```

```r
panel.fun = function(x, y) {
  xlim = get.cell.meta.data("xlim")
  ylim = get.cell.meta.data("ylim")
  theta = mean(get.cell.meta.data("xplot")) %% 360
  sector.index = get.cell.meta.data("sector.index")
  if(theta < 90 || theta > 270) {
    text.facing = "clockwise"
    text.adj = c(0, 0.5)
  } else {
    text.facing = "reverse.clockwise"
    text.adj = c(1, 0.5)
  }
  circos.text(mean(xlim), ylim[1],
              labels = paste(rep(sector.index, 8), collapse = ""),
              facing = text.facing, adj = text.adj, cex = 0.8)
}
circos.clear()
```
Description

Draw histogram in cells among a whole track

Usage

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)

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
- 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.

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.
circos.trackLines

References


Examples

```r
## Not run:
library(circlize)
par(mar = c(1, 1, 1, 1))
x = rnorm(2600)
factors = sample(letters, 2600, replace = TRUE)
circos.initialize(factors = factors, x = x)
circos.trackHist(factors = factors, x = x, track.height = 0.1,
    col = "#999999", border = "#999999")
circos.trackHist(factors = factors, x = x, force.ylim = FALSE,
    track.height = 0.1, col = "#999999", border = "#999999")
circos.trackHist(factors = factors, x = x, draw.density = TRUE,
    track.height = 0.1, col = "#999999", border = "#999999")
circos.trackHist(factors = factors, x = x, draw.density = TRUE,
    force.ylim = FALSE, track.height = 0.1, col = "#999999", border = "#999999")

circos.clear()

## End(Not run)
```

circos.trackLines  

Add lines to the plotting regions in a same track

Description

Add lines to the plotting regions in a same track

Usage

```r
circos.trackLines(factors, x, y, track.index = get.cell.meta.data("track.index"),
    col = "black", 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**: Factors which represent 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("1", "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.

References

Examples
# There is no example
NULL

circos.trackPlotRegion
Create plotting regions for a whole track

Description
Create plotting regions for a whole track

Usage
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 Factors which represent categories of data, if it is NULL, then it uses the whole 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 created it. Note the value for this argument should not exceed maximum track index plus 1.
track.height Height of the track. It is the percentage to the radius of the unit circles. 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 pretends to be a high-level plotting function, which means, you must first call this function to create plotting regions, then those low-level graphical function such as circos.points, circos.lines can be applied.

It has two different usages. First, it can create a complete track which among several sectors. Because currently it does not support creating single cell since it will make the layout disordered, this is the only way to create plotting regions.

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 limitation for values on x-axis has already been defined by circos.initialize, only limitation for values on y-axis should be specified in this function. The x argument is only used if you set panel.fun. There are two ways to identify the limitation 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 has overlap with other tracks, there will be an error.

panelNfun provides a convenient way to add graphics in each cell when initializing the tracks. The self-defined function need two arguments: x and y which correspond to the data points in the current cell. circosNtrackPlotRegion creates plotting regions one by one on the track and panelNfun 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.

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 latest created cell.

Second, it can update a already-created track if the index for the track is specified. If the index is one larger than the largest track index, it in fact creates the new track. 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.

References


Examples

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

---

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"))
```

Arguments

- `factors`: Factors which represent 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
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`.

References


Examples

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

circos.trackText  

---

draw trackText

Description

Draw text in cells among the whole track

Usage

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 = "black", font = par("font"))

Arguments

- **factors**: Factors which represent 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`.

References


Examples

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

circos.update  Create plotting regions for a whole track

Description

Create plotting regions for a whole track

Usage

```r
circos.update(...)
```

Arguments

...  pass to `circos.updatePlotRegion`

Details

shortcut function of `circos.updatePlotRegion`.

Examples

```r
# 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

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sector.index</td>
<td>Index for the sector</td>
</tr>
<tr>
<td>track.index</td>
<td>Index for the track</td>
</tr>
<tr>
<td>bg.col</td>
<td>Background color for the plotting region</td>
</tr>
<tr>
<td>bg.border</td>
<td>Color for the border of the plotting region</td>
</tr>
<tr>
<td>bg.lty</td>
<td>Line style for the border of the plotting region</td>
</tr>
<tr>
<td>bg.lwd</td>
<td>Line width for the border of the plotting region</td>
</tr>
</tbody>
</table>

Details

You can update an existed cell by this function by erasing all the graphics. But the xlim and ylim inside the cell still remains unchanged.

Note if you use `circos.trackPlotRegion` to update an already created track, you can re-define ylim in these cells.

References


Examples

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

**Description**

Draw x-axis

**Usage**

```
circos.xaxis(...)
```

**Arguments**

`...` all pass to `circos.axis`

**Examples**

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

---

circos.yaxis  

**Description**

Draw y-axis

**Usage**

```
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 = 0.5, lwd = par("lwd"))
```

**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. Also, the exceeding part would be trimmed automatically.
- `tick` Whether to draw ticks.
- `sector.index` Index for the sector
### 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

```r
circos.par()

# There is no example

NULL
```

---

### col2value

**Transform colors to values**

**Description**

Transform colors to values

**Usage**

```r
col2value(r, g, b, col_fun)
```

**Arguments**

- `r` red channel in sRGB color space, value should be between 0 and 1. It can also be a three-column matrix.
- `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.

**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)

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

---

<table>
<thead>
<tr>
<th>colorRamp2</th>
<th>Color interpolation</th>
</tr>
</thead>
</table>

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

Value

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

References

### Examples

```r
library(circlize)
col_fun = colorRamp2(c(-1, 0, 1), c("green", "black", "red"))
col_fun(seq(-2, 2, by = 0.5))

# map colors to p-values
col_fun = colorRamp2(c(log10(0.0001), log10(0.05), log10(1)), c("green", "white", "red"))
col_fun(log10(c(0.000001, 0.0012, 0.012, 0.2)))

# compare different color space
space = c("RGB", "HSV", "LAB", "XYZ", "sRGB", "LUV")

par(xpd = NA)
plot(NULL, xlim = c(-1, 1), ylim = c(0, length(space)+1), type = "n")
for(i in seq_along(space)) {
  f = colorRamp2(c(-1, 0, 1), c("green", "black", "red"), space = space[i])
  x = seq(-1, 1, length = 200)
  rect(x-1/200, i-0.5, x+1/200, i+0.5, col = f(x), border = NA)
  text(1, i, space[i], adj = c(-0.2, 0.5))
}
par(xpd = FALSE)
```

---

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](http://circos.ca/tutorials/course/slides/session-2.pdf), page 42.

### References

Examples

```r
## Not run:
 cytoband = read.cytoband()
 cytoband.col(cytoband$df[[5]])

## End(Not run)
```

---

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.

References

Examples

```r
## Not run:
library(circlize)
par(mar = c(1, 1, 1, 1))
plot(c(-1, 1), c(-1, 1), type = "n", axes = FALSE, ann = FALSE)
draw.sector(20, 0)
draw.sector(30, 60, rou1 = 0.8, rou2 = 0.5, clock.wise = FALSE, col = "#FF0000")
draw.sector(350, 1000, col = "#00FF00", 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 = "#0000FF")

par(mar = c(1, 1, 1, 1))
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 = "#FF0040")
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 = "#00FF00")
draw.sector(get.cell.meta.data("cell.start.degree", sector.index = "e"),
            get.cell.meta.data("cell.end.degree", sector.index = "f"),
            get.cell.meta.data("cell.top.radius", track.index = 2),
            get.cell.meta.data("cell.bottom.radius", track.index = 3),
            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()
## End(Not run)
```

Description

Generate random genomic data
Usage

\[
generateRandomBed(nr = 10000, nc = 1, fun = function(k) rnorm(k, 0, 0.5),
\text{species} = \text{NULL})
\]

Arguments

nr  Number of rows
nc  Number of numeric columns / value columns
fun Function for generating random values
species species, pass to \texttt{read.cytopand}

Details

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 \texttt{nr}.

References


Examples

\[
\begin{align*}
\text{bed} &= \text{generateRandomBed}() \\
\text{bed} &= \text{generateRandomBed}(nr = 200, nc = 4) \\
\text{bed} &= \text{generateRandomBed}(\text{fun} = \text{function}(k) \text{runif}(k))
\end{align*}
\]

\[
\begin{array}{ll}
\text{genomicDensity} & \text{Calculate genomic region density} \\
\end{array}
\]

Description

Calculate genomic region density

Usage

\[
\text{genomicDensity}(\text{region}, \text{window.size} = 10000000, \text{overlap} = \text{TRUE})
\]

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
overlap  Whether two neighbouring windows have half overlap
**get.all.sector.index**

**Details**

It calculates 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 percent of overlapping. And if the input is a bed-format data frame, there will be an additionally chromosome name column.

**References**


**Examples**

```r
## Not run:
bed = generateRandomBed()
bed = subset(bed, chr == "chr1")
genomicDensity(bed[2:3])

## End(Not run)
```

---

**get.all.sector.index**  
*Get index for all sectors*

**Description**

Get index for all sectors

**Usage**

get.all.sector.index()

**Details**

Simple function returning a vector of all sector index.

**References**

get.all.track.index

Examples

## Not run:
library(circlize)
factors = letters[1:4]
circos.initialize(factors, xlim = c(0, 1))
circos.trackPlotRegion(ylim = c(0, 1))
get.all.sector.index()
circos.clear()

## End(Not run)

---

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

Description

Get index for all tracks

Usage

get.all.track.index()

Details

Simple function returning a vector of all track index.

Examples

## Not run:
library(circlize)
factors = letters[1:4]
circos.initialize(factors, xlim = c(0, 1))
circos.trackPlotRegion(ylim = c(0, 1))
circos.trackPlotRegion(ylim = c(0, 1))
circos.trackPlotRegion(ylim = c(0, 1))
get.all.track.index()
circos.clear()

## End(Not run)
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.trackPlotRegion to get detailed information of the current cell.
get.current.chromosome

References


Examples

```r
## Not run:
library(circlize)
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()

## End(Not run)
```

---

get.current.chromosome

*Get current chromosome name*

Description

Get current chromosome name

Usage

```r
get.current.chromosome()
```

Details

The function is a simple wrapper of `get.cell.meta.data("sector.index")` and should only be put inside `panel.fun` when using `circos.genomicTrackPlotRegion`.

References


Examples

```r
## Not run:
library(circlize)
circos.initializeWithIdeogram()
circos.genomicTrackPlotRegion(ylim = c(0, 1), panel.fun = function(region, value, ...) {
  print(get.current.chromosome())
})
circos.clear()
```
getI

## End(Not run)

<table>
<thead>
<tr>
<th>getI</th>
<th>Which data that <code>panel.fun</code> is using</th>
</tr>
</thead>
</table>

**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.

**References**


**Examples**

```r
# There is no example
NULL
```
highlight.chromosome  Highlight chromosomes

Description

Highlight chromosomes

Usage

highlight.chromosome(...)

Arguments

...  pass to highlight.sector

Details

This is only a shortcut function of highlight.sector.

References


Examples

```r
## Not run:

par(mar = c(1.5, 1.5, 1.5, 1.5))
# highlight
circos.par("track.height" = 0.1, cell.padding = c(0, 0, 0, 0))
circos.initializeWithIdeogram(plotType = c("axis", "labels"))

circularTrackPlotRegion(bed, panel.fun = function(region, value, ...) {
  circos.genomicPoints(region, value, pch = 16, cex = 0.5, ...)
})

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

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

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

))

highlight.chromosome("chr1", col = "#FF000040", padding = c(0.05, 0.05, 0.15, 0.05))
highlight.chromosome("chr3", col = NA, border = "red", lwd = 2,
   padding = c(0.05, 0.05, 0.15, 0.05))
highlight.chromosome("chr5", col = "#0000FF40", track.index = c(2, 4, 5))
highlight.chromosome("chr7", col = NA, border = "green", lwd = 2,
   track.index = c(2, 4, 5))
circos.clear()

## End(Not run)

---

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
- `text.col`: Color for the text
- `...`: Pass to `circos.text`
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`.

References


Examples

```r
## Not run:
library(circlize)

par(mfrow = c(2, 1))
par(mar = c(1, 1, 1, 1))
### rect matrix
circos.par(cell.padding = c(0, 0, 0, 0))
circos.initializeWithIdeogram()

bed = generateRandomBed(nr = 100, nc = 4)

circos.genomicPosTransformLines(bed, posTransform = posTransform.default, horizontalline = "top", track.height = 0.1)

f = colorRamp2(breaks = c(-1, 0, 1), colors = c("green", "black", "red"))
circos.genomicTrackPlotRegion(bed, stack = TRUE, panel.fun = function(region, value, ...) {
  circos.genomicRect(region, value, col = f(value[[1]]), border = f(value[[1]])), posTransform = posTransform.default, ...)
}, bg.border = NA)

circos.clear()

circos.par(cell.padding = c(0, 0, 0, 0))
circos.initializeWithIdeogram(plotType = NULL)

bed = generateRandomBed(nr = 20, nc = 4)

circos.genomicTrackPlotRegion(bed, ylim = c(0, 1), panel.fun = function(region, value, ...) {
  circos.genomicText(region, value, y = 0, adj = c(1, 0.5), labels = "gene", facing = "reverse.clockwise", posTransform = posTransform.default)
}, bg.border = NA)

circos.genomicPosTransformLines(bed, posTransform = posTransform.default, horizontalline = "bottom", direction = "outside", track.height = 0.1)

cytoband = read.cytoband()$df
circos.genomicTrackPlotRegion(cytoband, stack = TRUE, panel.fun = function(region, value, ...) {
```
postransformNtext

```r
circos.genomicRect(region, value, col = cytoband.col(value$V5), border = NA, ...)
circos.clear()
```

### End(Not run)

---

**posTransform.text**  
*Genomic position transformation function specifically for text*

**Description**

Genomic position transformation function specifically for text

**Usage**

```r
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, ...)
```

**Arguments**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>region</td>
<td>Genomic positions at a single chromosome. It is a data frame with two columns which are start position and end position.</td>
</tr>
<tr>
<td>y</td>
<td>positions of texts</td>
</tr>
<tr>
<td>labels</td>
<td>text labels</td>
</tr>
<tr>
<td>cex</td>
<td>text size</td>
</tr>
<tr>
<td>font</td>
<td>text font style</td>
</tr>
<tr>
<td>sector.index</td>
<td>sector index</td>
</tr>
<tr>
<td>track.index</td>
<td>track index</td>
</tr>
<tr>
<td>padding</td>
<td>padding of text</td>
</tr>
<tr>
<td>...</td>
<td>other arguments</td>
</tr>
</tbody>
</table>

**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

```r
## Not run:

op = par(no.readonly = TRUE)
set.seed(123458)
par(mfrow = c(2, 2))
par(mar = c(1, 1, 1, 1))

bed = generateRandomBed(nr = 400, fun = function(k) rep("text", k))
bed = bed[-9:13,]

# circos.par("start.degree" = 90, canvas xlim = c(0, 1), canvas ylim = c(0, 1),
gap.degree = 270, cell.padding = c(0, 0, 0, 0), track.margin = c(0.005, 0.005))
circos.initializeWithIdeogram(plotType = c("axis"), chromosome.index = "chr1")
circos.genomicTrackPlotRegion(bed, ylim = c(0, 1),
   panel.fun = function(region, value, ...) {
      circos.genomicText(region, value, y = 0, labels.column = 1,
         facing = "clockwise", adj = c(0, 0.5),
         postTransform = posTransform.text, cex = 0.8, niceFacing = F)
   }, track.height = 0.1, bg.border = NA)
i_track = get.cell.meta.data("track.index")

circos.genomicPosTransformLines(bed,
   posTransform = function(region, value)
      posTransform.text(region, y = 0, labels = value[[1]],
         cex = 0.8, track.index = i_track),
   direction = "outside")

circos.genomicTrackPlotRegion(bed, ylim = c(0, 1),
   panel.fun = function(region, value, ...) {
      circos.points((region[[1]] + region[[2]])/2, rep(0.5, nrow(region)), pch = 16)
   }, track.height = 0.02, bg.border = NA)
circos.clear()

text(0, 0.05, "posTransform.text\ndirection = 'outside'", adj = c(0, 0))

# circos.par("start.degree" = 90, canvas.xlim = c(0, 1), canvas ylim = c(0, 1),
gap.degree = 270, cell.padding = c(0, 0, 0, 0), track.margin = c(0.005, 0.005))
circos.initializeWithIdeogram(plotType = c("axis"), chromosome.index = "chr1")
circos.genomicTrackPlotRegion(bed, ylim = c(0, 1),
   panel.fun = function(region, value, ...) {
      circos.genomicText(region, value, y = 0, labels.column = 1,
         facing = "clockwise", adj = c(0, 0.5),
         postTransform = posTransform.default, cex = 0.8, niceFacing = F)
   }, track.height = 0.1, bg.border = NA)
i_track = get.cell.meta.data("track.index")
```
```r
circos.genomicPosTransformLines(bed, posTransform = posTransform.default, 
direction = "outside")

circos.genomicTrackPlotRegion(bed, ylim = c(0,1), 
panel.fun = function(region, value, ...) { 
circos.points( (region[[1]] + region[[2]])/2, rep(0.5, nrow(region)), pch = 16) 
}, track.height = 0.02, bg.border = NA)

circos.clear()
text(0, 0.05, "posTransform.default\ndirection = 'outside'", adj = c(0, 0))

#---------------------------------------------
circos.par("start.degree" = 90, canvas.xlim = c(0,1), canvas ylim = c(0,1), 
gap.degree = 270, cell.padding = c(0, 0, 0, 0), track.margin = c(0.005, 0.005))
circos.initializeWithIdeogram(plotType = c("axis"), chromosome.index = "chr1")
circos.par(cell.padding = c(0, 0, 0, 0))
circos.genomicTrackPlotRegion(bed, ylim = c(0,1), 
panel.fun = function(region, value, ...) { 
circos.points( (region[[1]] + region[[2]])/2, rep(0.5, nrow(region)), pch = 16) 
}, track.height = 0.02, bg.border = NA)

circos.genomicTrackPlotRegion(bed, ylim = c(0,1), track.height = 0.1, bg.border = NA)
i_track = get.cell.meta.data("track.index")

circos.genomicTrackPlotRegion(bed, ylim = c(0,1), 
panel.fun = function(region, value, ...) { 
circos.genomicText(region, value, y = 1, labels.column = 1, 
   facing = "clockwise", adj = c(1, 0.5), 
   posTransform = posTransform.text, cex = 0.8, niceFacing = F) 
}, track.height = 0.1, bg.border = NA)

circos.genomicPosTransformLines(bed, 
posTransform = function(region, value) 
posTransform.text(region, y = 1, labels = value[[1]], 
cex = 0.8, track.index = i_track+1), 
direction = "inside", track.index = i_track)

circos.clear()
text(0, 0.05, "posTransform.text\ndirection = 'inside'", adj = c(0, 0))

#---------------------------------------------
circos.par("start.degree" = 90, canvas.xlim = c(0,1), canvas.ylim = c(0,1), 
gap.degree = 270, cell.padding = c(0, 0, 0, 0), track.margin = c(0.005, 0.005))
circos.initializeWithIdeogram(plotType = c("axis"), chromosome.index = "chr1")
circos.par(cell.padding = c(0, 0, 0, 0))
circos.genomicTrackPlotRegion(bed, ylim = c(0,1), 
panel.fun = function(region, value, ...) { 
circos.points( (region[[1]] + region[[2]])/2, rep(0.5, nrow(region)), pch = 16) 
}, track.height = 0.02, bg.border = NA)

circos.genomicTrackPlotRegion(bed, ylim = c(0,1), track.height = 0.1, bg.border = NA)
i_track = get.cell.meta.data("track.index")
```
Calculate inter-distance of genomic regions

Description

Calculate inter-distance of genomic regions

Usage

rainfallTransform(region, mode = c("min", "max", "mean"))

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.

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.
References


Examples

```r
## Not run:
bed = generateRandomBed()
bed = subset(bed, chr == "chr1")
rainfallTransform(bed[2:3])

## End(Not run)
```

---

**rand_color**

generate random colors

**Description**

generate random colors

**Usage**

```r
rand_color(n = 1, transparency = 0)
```

**Arguments**

- `n`: number of colors
- `transparency`: transparency, numeric value between 0 and 1

**Value**

a vector of colors

**Examples**

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

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

**Usage**

```r
read.chromInfo(chromInfo = paste0(system.file(package = "circlize"),
"/extdata/chromInfo.txt"), species = NULL, chromosome.index = NULL, 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 re-set chromosome orders.
- `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 cytoband data from [http://hgdownload.cse.ucsc.edu/goldenpath/hg19/database/chromInfo.txt.gz](http://hgdownload.cse.ucsc.edu/goldenpath/hg19/database/chromInfo.txt.gz)

If sort.chr is not set and chromosome.index is not specified, there would be several circumstances when determining the order of chromosomes. Assuming chromosome is the first column in the chromInfo data frame, then, if chromInfo is defined as a file path, or species is set, the order of chromosomes is unique(chromosome) which is read from the file; If chromInfo is set as a data frame and the first column is a factor, the order of chromosomes is levels(chromosome): If chromInfo is a data frame and the first column is just a character vector, the order of chromosomes is unique(chromosome). Please not this concept is really important since the order of chromosomes will be used to control the order of sectors when initializing the circos plot.

**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

# There is no example
NULL

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

Description

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

Usage

read.cytoband(cytoband = paste0(system.file(package = "circlize"),
"/extdata/cytoBand.txt"), species = NULL, chromosome.index = NULL, 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 or chromInfo.txt.gz from UCSC website automatically.

chromosome.index  subset of chromosomes, also used to re-set chromosome orders.

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 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 for the cytoband data from http://hgdownload.cse.ucsc.edu/goldenpath/hg19/database/cytoBand.txt.gz

If sort.chr is not set and chromosome.index is not specified, there would be several circumstances when determining the order of chromosomes. Assuming chromosome is the first column in the cytoband data frame, then, if cytoband is defined as a file path, or species is set, the order of chromosomes is unique(chromosome) which is read from the file; If cytoband is set as a data frame and the first column is a factor, the order of chromosomes is levels(chromosome); If cytoband is a data frame and the first column is just a character vector, the order of chromosomes is unique(chromosome). Please not this concept is really important since the order of chromosomes will be used to control the order of sectors when initializing the circos plot.
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`

References


Examples

```r
## Not run:
cytoband = read.cytoband(species = "hg19")
cytoband = read.cytoband(species = "mm10")
## End(Not run)
```

---

**reverse.circlize** Return the coordinate in data coordinate system

Description

Return the coordinate in data coordinate system

Usage

```r
reverse.circlize(theta, rou, sector.index = get.current.sector.index(),
                   track.index = get.current.track.index())
```

Arguments

- **theta**: measured by degree
- **rou**: distance to the circle center (radius)
- **sector.index**: Index for the sector
- **track.index**: Index for the track

Details

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

Value

A matrix with two columns (x and y)
Examples

```r
## Not run:
library(circlize)
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()

## End(Not run)
```

---

**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

```r
show.index()
```

Details

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

References


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

# There is no example
NULL
Index

adjacencyList2Matrix, 5
Arrowhead, 9, 12, 43
chordDiagram, 4, 5
chordDiagramFromDataFrame, 4, 6, 7, 12
chordDiagramFromMatrix, 4, 6, 10
circlize, 12, 83
circlize-package, 3
circos.axis, 3, 13, 60
circos.clear, 4, 15
circos.dendrogram, 16
circos.genomicDensity, 4, 17
circos.genomicInitialize, 4, 19, 38, 39
circos.genomicLines, 4, 20
circos.genomicLink, 4, 22
circos.genomicPoints, 4, 24
circos.genomicPostTransformLines, 26, 75
circos.genomicRainfall, 4, 28
circos.genomicRect, 4, 29
circos.genomicText, 4, 31
circos.genomicTrack, 33
circos.genomicTrackPlotRegion, 4, 21, 24, 30, 32, 33, 34, 70, 71
circos.info, 4, 36, 74, 84
circos.initialize, 16, 19, 37, 39, 45, 55
circos.initializeWithIdeogram, 4, 38
circos.lines, 3, 18, 21, 40, 54, 55
circos.link, 3, 9, 11, 12, 23, 42
circos.par, 4, 44, 61
circos.points, 3, 24, 46, 55, 57
circospolygon, 3, 47
circos.rect, 3, 29, 32, 48, 48
circos.segments, 49
circos.text, 3, 13, 32, 49, 58, 73
circos.track, 51
circos.trackHist, 52
circos.trackLines, 3, 53
circos.trackPlotRegion, 4, 18, 26, 28, 34, 35, 51, 54, 56, 59, 69
circos.trackPoints, 3, 56
circos.trackText, 3, 57
circos.update, 58
circos.updatePlotRegion, 4, 58, 59
circos.xaxis, 60
circos.yaxis, 60
col2value, 61
colorRamp2, 8, 10, 61, 62
cytoband.col, 63
dendrogram, 16
drawsector, 64, 74

generateRandomBed, 65
genomicDensity, 18, 66
get.all.sector.index, 67
get.all.track.index, 68
get.cell.meta.data, 69
get.current.chromosome, 70
getI, 35, 71
highlight.chromosome, 72
highlight.sector, 72, 73
hist, 52

LAB, 62
lines, 40, 49, 54
par, 45, 46
plot, 37
points, 46
polygon, 47, 48
postTransform.default, 21, 24, 26, 29, 32, 74
postTransform.text, 32, 76
rainfallTransform, 79
rand_color, 80
read.chromInfo, 38, 81
read.cytoband, 38, 39, 66, 82
rect, 48
reverse.circlize, 83
INDEX

show.index, 84
smartAlign, 85
sRGB, 61

text, 50