Package ‘spiralize’

July 8, 2021

<table>
<thead>
<tr>
<th>Type</th>
<th>Package</th>
</tr>
</thead>
<tbody>
<tr>
<td>Title</td>
<td>Visualize Data on Spirals</td>
</tr>
<tr>
<td>Version</td>
<td>1.0.2</td>
</tr>
<tr>
<td>Date</td>
<td>2021-07-03</td>
</tr>
<tr>
<td>Depends</td>
<td>R (&gt;= 3.6.0), grid</td>
</tr>
<tr>
<td>Imports</td>
<td>GlobalOptions (&gt;= 0.1.1), GetoptLong (&gt;= 0.1.8), circlize, stats, methods, grDevices, ComplexHeatmap, lubridate, utils</td>
</tr>
<tr>
<td>Suggests</td>
<td>knitr, rmarkdown, grImport, grImport2, jpeg, png, tiff, ape, cranlogs, cowplot, dendextend, bezier, magick</td>
</tr>
<tr>
<td>Description</td>
<td>It visualizes data along an Archimedean spiral [<a href="https://en.wikipedia.org/wiki/Archimedean_spiral">https://en.wikipedia.org/wiki/Archimedean_spiral</a>]. It has two major advantages for visualization: 1. It is able to visualize data with very long axis with high resolution. 2. It is efficient for time series data to reveal periodic patterns.</td>
</tr>
<tr>
<td>VignetteBuilder</td>
<td>knitr</td>
</tr>
<tr>
<td>URL</td>
<td><a href="https://github.com/jokergoo/spiralize">https://github.com/jokergoo/spiralize</a></td>
</tr>
<tr>
<td>License</td>
<td>MIT + file LICENSE</td>
</tr>
<tr>
<td>NeedsCompilation</td>
<td>no</td>
</tr>
<tr>
<td>Author</td>
<td>Zuguang Gu [aut, cre] [<a href="https://orcid.org/0000-0002-7395-8709">https://orcid.org/0000-0002-7395-8709</a>]</td>
</tr>
<tr>
<td>Maintainer</td>
<td>Zuguang Gu <a href="mailto:z.gu@dkfz.de">z.gu@dkfz.de</a></td>
</tr>
<tr>
<td>Repository</td>
<td>CRAN</td>
</tr>
<tr>
<td>Date/Publication</td>
<td>2021-07-08 21:40:02 UTC</td>
</tr>
</tbody>
</table>

\textbf{R topics documented:}

- `cartesian_to_xy` ................................................................. 2
- `current_spiral` ............................................................... 3
- `current_spiral_vp` ......................................................... 4
- `current_track_index` ....................................................... 5
- `get_track_data` ............................................................. 5
cartesian_to_xy

Convert canvas coordinates to the data coordinates

Description

Convert canvas coordinates to the data coordinates
Usage

cartesian_to_xy(x, y, track_index = current_track_index())

Arguments

x X-locations of the data points in canvas coordinates.
y Y-locations of the data points in canvas coordinates.
track_index Index of the track.

Details

The data points are assigned to the nearest inner loops. Denote the a data point has a coordinate (r, theta) in the polar coordinate system, r_k and r_k+1 are the radius of the two loops at theta + 2*pi*a and theta + 2*pi*(a+1) that below and above the data point, the data point is assigned to the loop k.

Value

A data frame with two columns: x and y.

Examples

x = runif(100, -5, 5)
y = runif(100, -5, 5)
spiral_initialize()
spiral_track()
df = cartesian_to_xy(x, y)
# directly draw in the viewport
grid.points(x, y, default.units="native")
# check whether the converted xy are correct (should overlap to the previous points)
spiral_points(df$x, df$y, pch = 16)

---

current_spiral Get current spiral object

Description

Get current spiral object

Usage

current_spiral()
Details

The returned value is an object of spiral reference class. The following methods might be useful:

$\text{curve}()$: It returns the radius for given angles (in radians).

$\text{spiral\_length}()$: It returns the length of the spiral (from the origin) for a given angle (in radians), thus if you want to get the length of a spiral segment, it will be $\text{spiral\_length(\theta_2)} - \text{spiral\_length(\theta_1)}$ where $\text{spiral}$ is the spiral object.

Also there are the following meta-data for the current spiral (assume the object is named $s$):

$s$\text{\$xlim}:
Data range.
$s$\text{\$xrange}:
$s$\text{\$xlim[2]} - s$\text{\$xlim[1]}
$s$\text{\$theta\_lim}:
The corresponding range of theta
$s$\text{\$theta\_range}:
$s$\text{\$theta\_lim[2]} - s$\text{\$theta\_lim[1]}
$s$\text{\$spiral\_length\_lim}:
The corresponding range of spiral length
$s$\text{\$spiral\_length\_range}:
$s$\text{\$spiral\_length\_lim[2]} - s$\text{\$spiral\_length\_lim[1]}
$s$\text{\$max\_radius}:
Radius at $s$\text{\$theta\_lim[2]}

Value

A spiral object.

Examples

`spiral\_initialize()`
`s = current\_spiral()
`s$curve(2*pi*2)
`s$spiral\_length(2*pi*2)`

---

`current\_spiral\_vp`  
Viewport name of current spiral

Description

Viewport name of current spiral

Usage

`current\_spiral\_vp()`

Value

A string of the viewport name.

Examples

# There is no example
NULL
**current_track_index**

**Description**
Current track index

**Usage**
current_track_index()

**Value**
An integer of the index of the current track.

**Examples**
# There is no example
NULL

---

**get_track_data**

Get meta-data of a track

**Description**
Get meta-data of a track

**Usage**
get_track_data(field, track_index = current_track_index())

**Arguments**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>field</td>
<td>Name, see Details section.</td>
</tr>
<tr>
<td>track_index</td>
<td>Which track?</td>
</tr>
</tbody>
</table>

**Details**
There are following fields that can be retrieved for a given track:

- ymin Lower boundary of the data.
- ymax Upper boundary of the data.
- ycenter \((ymin + ymax)/2\)
- ylim \(c(ylim, ymax)\)
• yrange ymax -ymin
• height Height of the track, measured as the fraction of the distance between two neighbouring circles.

It is more suggested to directly use TRACK_META to retrieve meta data for the current track.

Value

A numeric value of the corresponding field.

Examples

# There is no example
NULL

horizon_legend

Legend for the horizon chart

Description

Legend for the horizon chart

Usage

horizon_legend(lt, title = "", format = "%.2f", template = "[%{x1}, %{x2}]", ...)

Arguments

lt The object returned by spiral_horizon.
title Title of the legend.
format Number format of the legend labels.
template Template to construct the labels.
... Pass to Legend.

Value

A Legend object.

Examples

# There is no example
NULL
is_in_track

Test whether points are in a track

Description
Test whether points are in a track

Usage
is_in_track(x, y, track_index = current_track_index())

Arguments
- x: X-location of data points.
- y: Y-location of data points.
- track_index: Index of track.

Value
A logical vector.

Examples
# There is no example
NULL

names.TRACK_META

Names of all supported meta data

Description
Names of all supported meta data

Usage
## S3 method for class 'TRACK_META'
names(x)

Arguments
- x: Always use TRACK_META.

Value
A vector of characters.
Examples

names(TRACK_META)

<table>
<thead>
<tr>
<th>n_tracks</th>
<th>Number of tracks</th>
</tr>
</thead>
</table>

Description

Number of tracks

Usage

n_tracks()

Value

An integer of the number of available tracks.

Examples

# There is no example
NULL

phylo_to_dendrogram

Convert a phylo object to a dendrogram object

Description

Convert a phylo object to a dendrogram object

Usage

phylo_to_dendrogram(obj, log = FALSE)

Arguments

obj A phylo object.

log Whether the height of the phylogenetic tree should be log-transformed (log10(x + 1)).
Details
The motivation is that phylogenetic tree may contain polytomies, which means at a certain node, there are more than two children branches. Available tools that do the conversion only support binary trees.

The returned dendrogram object is not in its standard format which means it can not be properly drawn by the plot.dendrogram function. However, you can still apply dendextend::cutree to the returned dendrogram object with no problem and the dendrogram can be properly drawn with the ComplexHeatmap package.

Value
A dendrogram object.

Examples
```r
require(ape)
data(bird.families)
d = phylo_to_dendrogram(bird.families)

require(ComplexHeatmap)
gird.dendrogram(d, test = TRUE)
```

polar_to_cartesian

Convert polar coordinates to cartesian coordinates

Description
Convert polar coordinates to cartesian coordinates

Usage
```
polar_to_cartesian(theta, r)
```

Arguments
- `theta` : Angles, in radians.
- `r` : Radius.

Value
A data frame with two columns: x and y.

Examples
```
# There is no example
NULL
```
set_current_track

Description
Set current track

Usage
set_current_track(track_index)

Arguments
track_index The index of the track.

Value
No value is returned.

---

print.TRACK META

Description
Print TRACK META

Usage
## S3 method for class 'TRACK META'
print(x, ...)

Arguments
x The TRACK META object.
... Additional parameters.

Value
No value is returned.

Examples
# There is no example
NULL
solve_theta_from_spiral_length

Examples

# There is no example
NULL

solve_theta_from_spiral_length

*Get theta from given spiral lengths*

Description

Get theta from given spiral lengths

Usage

solve_theta_from_spiral_length(len, interval = NULL, offset = 0)

Arguments

- **len**: A vector of spiral lengths.
- **interval**: Interval to search for the solution.
- **offset**: Offset of the spiral. In the general form: $r = a + r \theta$, offset is the value of $a$.

Details

The length of the spiral has a complicated form, see [https://downloads.imagej.net/fiji/snapshots/arc_length.pdf](https://downloads.imagej.net/fiji/snapshots/arc_length.pdf). Let's say the form is $l = f(\theta)$, `solve_theta_from_spiral_length` tries to find theta by a known $l$. It uses `uniroot` to search solutions.

Value

The theta value.

Examples

```r
spiral_initialize()
s = current_spiral()
theta = pi*seq(2, 3, length = 10)
len = s$spiral_length(theta)
solve_theta_from_spiral_length(len) # should be very similar as theta
```
spiral_arrow  

*Draw arrows in the spiral direction*

**Description**

Draw arrows in the spiral direction

**Usage**

```r
spiral_arrow(
  x1, x2,
  y = get_track_data("ycenter", track_index),
  width = get_track_data("yrange", track_index)/3,
  arrow_head_length = unit(4, "mm"),
  arrow_head_width = width*2,
  arrow_position = c("end", "start"),
  tail = c("normal", "point"),
  gp = gpar(),
  track_index = current_track_index())
```

**Arguments**

- `x1`  
  
  Start of the arrow.

- `x2`  
  
  End of the arrow.

- `y`  
  
  Y-location of the arrow.

- `width`  
  
  Width of the arrow. The value can be the one measured in the data coordinates or a `unit` object.

- `arrow_head_length`  
  
  Length of the arrow head.

- `arrow_head_width`  
  
  Width of the arrow head.

- `arrow_position`  
  
  Position of the arrow. If the value is "end", then the arrow head is drawn at \( x = x2 \). If the value is "start", then the arrow head is drawn at \( x = x1 \).

- `tail`  
  
  The shape of the arrow tail.

- `gp`  
  
  Graphics parameters.

- `track_index`  
  
  Index of the track.

**Value**

No value is returned.

**See Also**

Note `spiral_segments` also supports drawing line-based arrows.
**spiral_axis**

**Examples**

```r
spiral_initialize()
spiral_track()
spiral_arrow(0.3, 0.6, gp = gpar(fill = "red"))
spiral_arrow(0.8, 0.9, gp = gpar(fill = "blue"), tail = "point", arrow_position = "start")
```

---

**spiral_axis**  
*Draw axis along the spiral*

**Description**

Draw axis along the spiral

**Usage**

```r
spiral_axis(h = c("top", "bottom"), at = NULL, major_at = at, labels = TRUE, curved_labels = FALSE, minor_ticks = 4, major_ticks_length = unit(4, "bigpts"), minor_ticks_length = unit(2, "bigpts"), ticks_gp = gpar(), labels_gp = gpar(fontsize = 6), track_index = current_track_index())
```

**Arguments**

- **h**  
  Position of the axis. The value can be a character of "top" or "bottom".

- **at**  
  Breaks points on axis.

- **major_at**  
  Breaks points on axis. It is the same as `at`.

- **labels**  
  The corresponding labels for the break points.

- **curved_labels**  
  Whether are the labels are curved?

- **minor_ticks**  
  Number of minor ticks.

- **major_ticks_length**  
  Length of the major ticks. The value should be a `unit` object.

- **minor_ticks_length**  
  Length of the minor ticks. The value should be a `unit` object.

- **ticks_gp**  
  Graphics parameters for the ticks.

- **labels_gp**  
  Graphics parameters for the labels.

- **track_index**  
  Index of the track.

**Value**

No value is returned.
Examples

```r
spiral_initialize(); spiral_track()
spiral_axis()

# if the spiral is interpolated by the curve length
spiral_initialize(scale_by = "curve_length"); spiral_track()
spiral_axis()

spiral_initialize(xlim = c(0, 360*4), start = 360, end = 360*5); spiral_track()
spiral_axis(major_at = seq(0, 360*4, by = 30))

spiral_initialize(xlim = c(0, 12*4), start = 360, end = 360*5); spiral_track()
spiral_axis(major_at = seq(0, 12*4, by = 1), labels = c("", rep(month.name, 4)))
```

---

**spiral_bars**

Add bars to a track

**Description**

Add bars to a track

**Usage**

```r
spiral_bars(pos, value, baseline = get_track_data("ymin", track_index),
            bar_width = min(diff(pos)), gp = gpar(), track_index = current_track_index())
```

**Arguments**

- `pos` X-locations of the center of bars.
- `value` Height of bars. The value can be a simple numeric vector, or a matrix.
- `baseline` Baseline of the bars. Note it only works when `value` is a simple vector.
- `bar_width` Width of bars.
- `gp` Graphical parameters.
- `track_index` Index of the track.

**Value**

No value is returned.

**Examples**

```r
x = seq(1, 1000, by = 1) - 0.5
y = runif(1000)
spiral_initialize(xlim = c(0, 1000))
spiral_track(height = 0.8)
spiral_bars(x, y)
```
# a three-column matrix
y = matrix(runif(3*1000), ncol = 3)
y = y/rowSums(y)
spiral_initialize(xlim = c(0, 1000))
spiral_track(height = 0.8)
spiral_bars(x, y, gp = gpar(fill = 2:4, col = NA))

spiral_clear

Clear the spiral curve

Description

Clear the spiral curve

Usage

spiral_clear(check_vp = TRUE)

Arguments

check_vp Whether to check the viewport.

Details

It basically sets the internally spiral object to NULL, and reset all the global options.

Value

No value is returned.

Examples

# There is no example
NULL
spiral_dendrogram  
*Draw dendrogram*

**Description**

Draw dendrogram

**Usage**

```r
spiral_dendrogram(dend, gp = gpar(), track_index = current_track_index())
```

**Arguments**

- `dend`: A stats::dendrogram object.
- `gp`: Graphics parameters of the dendrogram edges.
- `track_index`: Index of the track.

**Details**

Note the dendrogram edges can be rendered with the `dendextend` package.

**Value**

Height of the dendrogram.

**Examples**

```r
k = 500
dend = as.dendrogram(hclust(dist(runif(k))))
spiral_initialize(xlim = c(0, k), start = 360, end = 360*3)
spiral_track(height = 0.8, background_gp = gpar(fill = "#EEEEEE", col = NA))
require(dendextend)
dend = color_branches(dend, k = 4)
spiral_initialize(xlim = c(0, k), start = 360, end = 360*3)
spiral_track(height = 0.8, background_gp = gpar(fill = "#EEEEEE", col = NA))
```
spiral_highlight

Highlight a section of the spiral

Description

Highlight a section of the spiral

Usage

spiral_highlight(x1, x2, type = c("rect", "line"), padding = unit(1, "mm"),
    line_side = c("inside", "outside"), line_width = unit(1, "pt"),
    gp = gpar(fill = "red"), track_index = current_track_index())

Arguments

x1 Start location of the highlighted section.
x2 End location of the highlighted section.
type Type of the highlighting. "rect" means drawing transparent rectangles covering the whole track. "line" means drawing annotation lines on top of the track or at the bottom of it.
padding When the highlight type is "rect", it controls the padding of the highlighted region. The value should be a unit object or a numeric value which is the fraction of the length of the highlighted section. The length can be one or two. Note it only extends in the radial direction.
line_side If the highlight type is "line", it controls which side of the track to draw the lines.
line_width Width of the annotation line. Value should be a unit object.
gp Graphics parameters.
track_index Index of the track.

Value

No value is returned.

Examples

spiral_initialize(); spiral_track()
spiral_highlight(0.4, 0.6)
spiral_highlight(0.1, 0.2, type = "line", gp = gpar(col = "blue"))
spiral_highlight(0.7, 0.8, type = "line", line_side = "outside")
**spiral_highlight_by_sector**

*Highlight a sector*

**Description**
Highlight a sector

**Usage**

```r
spiral_highlight_by_sector(x1, x2, x3 = NULL, x4 = NULL, padding = unit(1, "mm"), gp = gpar(fill = "red"))
```

**Arguments**

- **x1**
  Start location which determines the start of the sector.

- **x2**
  End location which determines the end of the sector. Note x2 should be larger than x1 and the angular difference between x1 and x2 should be smaller than a circle.

- **x3**
  Start location which determines the start of the sector on the upper border.

- **x4**
  End location which determines the end of the sector on the upper border.

- **padding**
  It controls the radial extension of the sector. The value should be a `unit` object with length one or two.

- **gp**
  Graphics parameters.

**Details**

x1 and x2 determine the position of the highlighted sector. If x3 and x4 are not set, the sector extends until the most outside loop. If x3 and x4 are set, they determine the outer border of the sector. In this case, if x3 and x4 are set, x3 should be larger than x2.

**Value**

No value is returned.

**Examples**

```r
spiral_initialize(xlim = c(0, 360*4), start = 360, end = 360*5)
spiral_track()
spiral_axis()
spiral_highlight_by_sector(36, 72)
spiral_highlight_by_sector(648, 684)
spiral_highlight_by_sector(216, 252, 936, 972, gp = gpar(fill = "blue"))
```
**spiral_horizon**

---

**spiral_horizon** \hspace{1cm} *Draw horizon chart along the spiral*

---

**Description**

Draw horizon chart along the spiral

**Usage**

```r
spiral_horizon(x, y, n_slices = 4, slice_size, pos_fill = "#D73027", neg_fill = "#313695",
use_bars = FALSE, bar_width = min(diff(x)),
negative_from_top = FALSE, track_index = current_track_index())
```

**Arguments**

- **x**: X-locations of the data points.
- **y**: Y-locations of the data points.
- **n_slices**: Number of slices.
- **slice_size**: Size of the slices. The final number of sizes is \(\text{ceiling}(\max(\text{abs}(y))/\text{slice_size})\).
- **pos_fill**: Colors for positive values.
- **neg_fill**: Colors for negative values.
- **use_bars**: Whether to use bars?
- **bar_width**: Width of bars.
- **negative_from_top**: Should negative distribution be drawn from the top?
- **track_index**: Index of the track.

**Details**

Since the track height is very small in the spiral, horizon chart visualization is a efficient way to visualize distribution-like graphics.

**Value**

A list of the following objects:

- a color mapping function for colors.
- a vector of intervals that split the data.
Examples

```r
df = readRDS(system.file("extdata", "global_temperature.rds", package = "spiralize"))
df = df[df$Source == "GCAG", ]
spiral_initialize_by_time(xlim = range(df$Date), unit_on_axis = "months", period = "year",
period_per_loop = 20, polar_lines_by = 360/20,
vp_param = list(x = unit(0, "npc"), just = "left"))
spiral_track()
spiral_horizon(df$Date, df$Mean, use_bar = TRUE)
```

---

**spiral_info**

Information of the current spiral

Description

Information of the current spiral

Usage

```r
spiral_info()
```

Details

It prints information of the current spiral.

Value

No value is returned.

Examples

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

---

**spiral_initialize**

Initialize the spiral

Description

Initialize the spiral
Usage

```r
spiral_initialize(xlim = c(0, 1), start = 360, end = 360*5,
   scale_by = c("angle", "curve_length"),
   flip = c("none", "vertical", "horizontal", "both"), reverse = FALSE,
   polar_lines = scale_by == "angle", polar_lines_by = 30,
   polar_lines_gp = gpar(col = "#808080", lty = 3),
   padding = unit(5, "mm"), newpage = TRUE, vp_param = list())
```

Arguments

- **xlim** Range on x-locations.
- **start** Start of the spiral, in degree. `start` and `end` should be positive and `start` should be smaller than `end`.
- **end** End of the spiral, in degree.
- **scale_by** How scales on x-axis are equally interpolated? The values can be one of "angle" and "curve_length". If the value is "angle", equal angle difference corresponds to equal difference of data. In this case, in outer loops, the scales are longer than in the inner loops, although the difference on the data are the same. If the value is "curve_length", equal curve length difference corresponds to the equal difference of the data.
- **flip** How to flip the spiral? By default, the spiral starts from the origin of the coordinate and grows reverseclockwisely. The argument controls the growing direction of the spiral.
- **reverse** By default, the most inside of the spiral corresponds to the lower boundary of x-location. Setting the value to FALSE can reverse the direction.
- **polar_lines** Whether draw the polar guiding lines.
- **polar_lines_by** Increment of the polar lines. Measured in degree.
- **polar_lines_gp** Graphics parameters for the polar lines.
- **padding** Padding of the plotting region. The value can be a unit of length of one to two.
- **newpage** Whether to apply `grid.newpage` before making the plot?
- **vp_param** A list of parameters sent to `viewport`.

Value

No value is returned.

Examples

```r
spiral_initialize(); spiral_track()
spiral_initialize(start = 180, end = 360+180); spiral_track()
spiral_initialize(flip = "vertical"); spiral_track()
spiral_initialize(flip = "horizontal"); spiral_track()
spiral_initialize(flip = "both"); spiral_track()
spiral_initialize(); spiral_track(); spiral_axis()
```
# the following example shows the difference of `scale_by` more clearly:
make_plot = function(scale_by) {
  n = 100
  require(circlize)
  col = circlize::colorRamp2(c(0, 0.5, 1), c("blue", "white", "red"))
  spiral_initialize(xlim = c(0, n), scale_by = scale_by)
  spiral_track(height = 0.9)
  x = runif(n)
  spiral_rect(1:n - 1, 0, 1:n, 1, gp = gpar(fill = col(x), col = NA))
}
make_plot("angle")
make_plot("curve_length")

spiral_initialize_by_gcoor

*Initialize the spiral with genomic coordinates*

**Description**

Initialize the spiral with genomic coordinates

**Usage**

```
spiral_initialize_by_gcoor(xlim, scale_by = "curve_length", ...)
```

**Arguments**

- `xlim` Range of the genomic coordinates.
- `scale_by` For genomic plot, axis is linearly scaled by the curve length.
- `...` All pass to `spiral_initialize`.

**Details**

It is basically the same as `spiral_initialize`. The only difference is the axis labels are automatically formatted for genomic coordinates.

**Value**

No value is returned.

**Examples**

```
spiral_initialize_by_gcoor(c(0, 1000000000))
spiral_track()
spiral_axis()
```
spiral_initialize_by_time

Initialize the spiral from time objects

Description

Initialize the spiral from time objects

Usage

spiral_initialize_by_time(xlim, start = NULL, end = NULL,
unit_on_axis = c("days", "months", "weeks", "hours", "mins", "secs"),
period = c("years", "months", "weeks", "days", "hours", "mins"),
period_per_loop = 1, polar_lines_by = NULL, verbose = TRUE, ...)

Arguments

xlim Range of the time. The value can be time object such as Date, POSIXlt or POSIXct. The value can also be characters and it is converted to time objects automatically.
start Start of the spiral, in degrees. By default it is automatically calculated.
end End of the spiral, in degrees. By default it is automatically calculated.
unit_on_axis Units on the axis.
period Which period to use?
period_per_loop How many periods to put in a loop?
polar_lines_by By default different value of polar_lines_by is set for different period. E.g. 360/7 is set if period is "weeks" or 360/24 is set if period is set to "hours".
verbose Whether to print messages?
... All pass to spiral_initialize.

Details

"start" and "end" are automatically calculated for different "unit_on_axis" and "period". For example, if "unit_on_axis" is "days" and "period" is "years", then the first day of each each year is always put on theta = 0 + 2*pi*k where k is the index of loops.

Value

No value is returned.
spiral_lines

Add lines to a track

Description

Add lines to a track

Usage

spiral_lines(x, y, type = "l", gp = gpar(),
  baseline = "bottom", area = FALSE, track_index = current_track_index())

Arguments

- **x**  
  X-locations of the data points.
- **y**  
  Y-locations of the data points.
- **type**  
  Type of the line. Value should be one of "l" and "h". When the value is "h", vertical lines (or radial lines if you consider the polar coordinates) relative to the baseline will be drawn.
- **gp**  
  Graphical parameters.
- **baseline**  
  Baseline used when type is "l" or area is TRUE.
- **area**  
  Whether to draw the area under the lines? Note gpar(fill = ...) controls the filled of the areas.
- **track_index**  
  Index of the track.

Value

No value is returned.
Examples

```r
x = sort(runif(1000))
y = runif(1000)
spiral_initialize()
spiral_track()
spiral_lines(x, y)

spiral_initialize()
spiral_track()
spiral_lines(x, y, type = "h")

spiral_initialize()
spiral_track()
spiral_lines(x, y, area = TRUE, gp = gpar(fill = "red", col = NA))
```

---

**Description**

Global options

**Usage**

`spiral_opt(..., RESET = FALSE, READ.ONLY = NULL, LOCAL = FALSE, ADD = FALSE)`

**Arguments**

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

**Details**

There are following global parameters:

- `min_segment_len` Minimal length of the segment that partitions a curve.

To access the value of an option: `spiral_opt$name` where name is the name of the option. To set a new value for an option: `spiral_opt$name = new_value`.

**Value**

A list of options.
spiral_phylo

Draw phylogenetic tree

Description

Draw phylogenetic tree

Usage

spiral_phylo(obj, gp = gpar(), log = FALSE, reverse = FALSE,
            group = NULL, group_col = NULL, track_index = current_track_index())

Arguments

obj  A phylo object.
gp  Graphics parameters of the tree edges.
log  Whether the height of the tree should be log-transformed (log10(x + 1))?
reverse  Whether the tree should be reversed?
group  A categorical variable for splitting the tree.
group_col  A named vector which contains group colors.
track_index  Index of the track.

Value

Height of the phylogenetic tree.

Examples

require(ape)
data(bird.families)
n = length(bird.families$tip.label)
spiral_initialize(xlim = c(0, n), start = 360, end = 360*3)
spiral_track(height = 0.8)
spiral_phylo(bird.families)
spiral_points

Description
Add points to a track

Usage
spiral_points(x, y, pch = 1, size = unit(0.4, "char"), gp = gpar(),
track_index = current_track_index())

Arguments
- x: X-locations of the data points.
- y: Y-locations of the data points.
- pch: Point type.
- size: Size of the points. Value should be a unit object.
- gp: Graphical parameters.
- track_index: Index of the track.

Value
No value is returned.

Examples
spiral_initialize()
spiral_track()
spiral_points(x = runif(1000), y = runif(1000))

spiral_polygon

Description
Add polygons to a track

Usage
spiral_polygon(x, y, id = NULL, gp = gpar(), track_index = current_track_index())
spiral_raster

Arguments

- **x**: X-locations of the data points.
- **y**: Y-locations of the data points.
- **id**: A numeric vector used to separate locations in x and y into multiple polygons.
- **gp**: Graphical parameters.
- **track_index**: Index of the track.

Details

Note the polygon must be closed, which means, the last data point should overlap to the first one.

Value

No value is returned.

Examples

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

spiral_raster  Add image to a track

Description

Add image to a track

Usage

```r
spiral_raster(x, y, image, width = NULL, height = NULL,
               facing = c("downward", "inside", "outside", "curved_inside", "curved_outside"),
               nice_facing = FALSE, scaling = 1, track_index = current_track_index())
```

Arguments

- **x**: X-locations of the center of the image.
- **y**: Y-locations of the center of the image.
- **image**: A vector of file paths of images. The format of the image is inferred from the suffix name of the image file. NA value or empty string means no image to drawn. Supported formats are png/svg/pdf/eps/jpeg/jpg/tiff.
- **width**: Width of the image. See Details.
- **height**: Height of the image. See Details.
- **facing**: Facing of the image.
- **nice_facing**: Whether to adjust the facing.
- **scaling**: Scaling factor when facing is set to "curved_inside" or "curved_outside".
- **track_index**: Index of the track.
spiral_rect

Details
When facing is set to one of "downward", "inside" and "outside", both of width and height should be unit objects. It is suggested to only set one of width and height, the other dimension will be automatically calculated from the aspect ratio of the image.

When facing is set to one of "curved_inside" and "curved_outside", the value can also be numeric, which are the values measured in the data coordinates. Note when the segment in the spiral that corresponds to width is very long, drawing the curved image will be very slow because each pixel is actually treated as a single rectangle.

Value
No value is returned.

Examples
image = system.file("extdata", "Rlogo.png", package = "circlize")
x = seq(0.1, 0.9, length = 10)

spiral_initialize()
spiral_track()
spiral_raster(x, 0.5, image)

spiral_initialize()
spiral_track()
spiral_raster(x, 0.5, image, facing = "inside")

spiral_rect
Add rectangles to a track

Description
Add rectangles to a track

Usage
spiral_rect(xleft, ybottom, xright, ytop, gp = gpar(),
track_index = current_track_index())

Arguments
xleft X-locations of the left bottom of the rectangles.
ybottom Y-locations of the left bottom of the rectangles.
xright X-locations of the right top of the rectangles.
ytop Y-locations of the right top of the rectangles.
gp Graphical parameters.
track_index Index of the track.
spiral_segments

Value

No value is returned.

Examples

# to simulate heatmap
n = 1000
require(circlize)
col = circlize::colorRamp2(c(0, 0.5, 1), c("blue", "white", "red"))
spiral_initialize(xlim = c(0, n))
spiral_track(height = 0.9)

x1 = runif(n)
spiral_rect(1:n - 1, 0, 1:n, 0.5, gp = gpar(fill = col(x1), col = NA))
x2 = runif(n)
spiral_rect(1:n - 1, 0.5, 1:n, 1, gp = gpar(fill = col(x2), col = NA))

spiral_segments  Add segments to a track

Description

Add segments to a track

Usage

spiral_segments(x0, y0, x1, y1, gp = gpar(), arrow = NULL,
track_index = current_track_index(), buffer = 10000)

Arguments

x0  X-locations of the start points of the segments.
y0  Y-locations of the start points of the segments.
x1  X-locations of the end points of the segments.
y1  Y-locations of the end points of the segments.
gp  Graphical parameters.
arrow  A arrow object.
track_index  Index of the track.
buffer  Number of segments to buffer.

Details

The segments on spiral are not straight lines while are more like curves. This means a spiral segment is formed by a list of real straight segments. If there are n1 spiral segments, then there will be n2 straight segments where n2 is normally much larger than n1. To speed up drawing the spiral segments, the locations of the "real" segments are filled to a temporary data frame with buffer rows, when the number of rows exceeds buffer, grid.segments is called to draw all the buffered segments.
spiral_text

Value

No value is returned.

Examples

```
n = 1000
x0 = runif(n)
y0 = runif(n)
x1 = x0 + runif(n, min = -0.01, max = 0.01)
y1 = 1 - y0

spiral_initialize(xlim = range(c(x0, x1)))
spiral_track()
spiral_segments(x0, y0, x1, y1, gp = gpar(col = circlize::rand_color(n)))
```

```
n = 100
x0 = runif(n)
y0 = runif(n)
x1 = x0 + runif(n, min = -0.01, max = 0.01)
y1 = 1 - y0

spiral_initialize(xlim = range(c(x0, x1)))
spiral_track()
spiral_segments(x0, y0, x1, y1, arrow = arrow(length = unit(2, "mm")),
                gp = gpar(col = circlize::rand_color(n, luminosity = "bright"), lwd = runif(n, 0.5, 3)))
```

---

**spiral_text** Add texts to a track

Description

Add texts to a track

Usage

```
spiral_text(x, y, text, offset = NULL, gp = gpar(),
            facing = c("downward", "inside", "outside", "clockwise", "reverse_clockwise",
                      "curved_inside", "curved_outside"),
            letter_spacing = 0,
            nice_facing = FALSE, just = "centre", hjust = NULL, vjust = NULL,
            track_index = current_track_index(), ...)
```

Arguments

- **x** X-locations of the texts.
- **y** Y-locations of the texts.
- **text** A vector of texts.
- **offset** Radial offset of the text. The value should be a unit object.
### GP
- **Graphical parameters.**

### Facing
- **Facing of the text.**

### Letter_spacing
- **Space between two letters.** The value is a fraction of the width of current letter. It only works for curved texts.

### Nice_facing
- If it is true, the facing will be automatically adjusted for texts which locate at different positions of the spiral. Note hjust and vjust will also be adjusted.

### Just
- **The justification of the text relative to (x, y).** The same setting as in `grid.text`.

### Hjust
- **Horizontal justification.** Value should be numeric. 0 means the left of the text and 1 means the right of the text.

### Vjust
- **Vertical justification.** Value should be numeric. 0 means the bottom of the text and 1 means the top of the text.

### Track_index
- **Index of the track.**

### ... (ellipsis)
- **Pass to `grid.text`**.

### Details

For the curved text, it only supports one-line text.

### Value

No value is returned.

### Examples

```r
x = seq(0.1, 0.9, length = 26)
text = strrep(letters, 6)
spiral_initialize(); spiral_track()
spiral_text(x, 0.5, text)

spiral_initialize(); spiral_track()
spiral_text(x, 0.5, text, facing = "inside")

spiral_initialize(); spiral_track()
spiral_text(x, 0.5, text, facing = "outside")

x = seq(0.1, 0.9, length = 10)
text = strrep(letters[1:10], 20)
spiral_initialize(); spiral_track()
spiral_text(x, 0.5, text, facing = "curved_inside")

spiral_initialize(); spiral_track()
spiral_text(x, 0.5, text, facing = "curvedOutside")
```
spiral_track

Add a new track or go to an existed track

Description

Add a new track or go to an existed track

Usage

spiral_track(ylim = c(0, 1), height = 0.8, background = TRUE,
    background_gp = gpar(col = NA, fill = "#EEEEEE"), reverse_y = FALSE,
    track_index = current_track_index() + 1)

Arguments

- **ylim**: Data range of the y-locations.
- **height**: Height of the track. The value can be the fraction of the distance of the two neighbour loops. The value can also be a `unit` object.
- **background**: Whether to draw the background of the track, i.e. border and filled color of background.
- **background_gp**: Graphics parameters of the background.
- **reverse_y**: Whether reverse the direction of y-axis.
- **track_index**: Index of the track.

Details

If the track is already existed, the function simply mark the track as the current track and does nothing else.

Value

No value is returned.

Examples

spiral_initialize()
spiral_track(height = 0.8)

spiral_initialize()
spiral_track(height = 0.4, background_gp = gpar(fill = "red"))
spiral_track(height = 0.2, background_gp = gpar(fill = "green"))
spiral_track(height = 0.1, background_gp = gpar(fill = "blue"))
**spiral_xaxis**  
*Draw axis along the spiral*

**Description**

Draw axis along the spiral

**Usage**

`spiral_xaxis(...)`

**Arguments**

...  
All pass to `spiral_axis`.

**Value**

No value is returned.

**Examples**

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

---

**spiral_yaxis**  
*Draw y-axis*

**Description**

Draw y-axis

**Usage**

```r
spiral_yaxis(side = c("both", "start", "end"), at = NULL, labels = TRUE,  
ticks_length = unit(2, "bigpts"),  
ticks_gp = gpar(), labels_gp = gpar(fontsize = 6),  
track_index = current_track_index())
```
Arguments

side On which side of the spiral the y-axis is drawn? "start" means the inside of the spiral and "end" means the outside of the spiral. Note if reverse was set to TRUE, then "start" corresponds to the most outside of the spiral.
at Break points.
lables Corresponding labels for the break points.
ticks_length Length of the tick. Value should be a unit object.
ticks_gp Graphics parameters for ticks.
lables_gp Graphics parameters for labels.
track_index Index of the track.

Value

No value is returned.

Examples

spiral_initialize(); spiral_track(height = 0.8)
spiral_yaxis("start")
spiral_yaxis("end", at = c(0, 0.25, 0.5, 0.75, 1), labels = letters[1:5])

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

TRACK_META Get meta data in the current track
-------------------------------------------------------------------

Description

Get meta data in the current track

Usage

TRACK_META

Details

The variable TRACK_META can only be used to get meta data from the "current" track. If the current track is not the one you want, you can first use set_current_track to set the current track.

Don’t directly use TRACK_META. The value of TRACK_META itself is meaningless. Always use in form of TRACK_META$\text{name}.

There are following meta data for the current track:

xlim: Data range on x-axis.
xmin: xlim[1].
xmax: xlim[2].
xcenter: mean(xlim).
theta_lim: Range of the angles on the spiral, measured in radians.
theta_min: theta_lim[1].
theta_max: theta_lim[2].
theta_center: mean(theta_lim).
ylim: Data range on y-axis.
ymin: ylim[1].
ymax: ylim[2].
ycenter: mean(ylim).
rel_height: Fraction of height of the track to the distance between two neighbouring loops.
abs_height: The height of the track, which is rel_height multiplied by the distance between two neighbouring loops.
track_index: Current track index.

Examples

# There is no example
NULL

---

**xy_to_cartesian**

Convert data coordinates to the canvas coordinates

**Description**

Convert data coordinates to the canvas coordinates

**Usage**

xy_to_cartesian(x, y, track_index = current_track_index())

**Arguments**

- **x**: X-locations of the data points.
- **y**: Y-locations of the data points.
- **track_index**: Index of the track.

**Details**

The canvas coordinates correspond to the "native" coordinates of the viewport where the graphics are to be drawn.  
Note different settings of flip and reverse in spiral_initialize affect the conversion.
### Description

Convert data coordinates to polar coordinates

### Usage

```r
xy_to_polar(x, y, track_index = current_track_index(), flip = TRUE)
```

### Arguments

- `x`: X-locations of the data points.
- `y`: Y-locations of the data points.
- `track_index`: Index of the track.
- `flip`: If it is FALSE, it returns theta for the original spiral (before flipping).

### Details

Note different settings of flip and reverse in `spiral_initialize` affect the conversion.

### Value

A data frame with two columns: theta (in radians) and r (the radius).

### Examples

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

Get meta data in the current track

### Usage

```r
## S3 method for class 'TRACK_META'
x$name
```

### Arguments

- `x`: Always use TRACK_META.
- `name`: Name of the meta name. For all supported names, type `names(TRACK_META)`.

### Details

The variable TRACK_META can only be used to get meta data from the "current" track. If the current track is not the one you want, you can first use `set_current_track` to set the current track.

There are following meta data for the current track:

- **xlim**: Data range on x-axis.
- **xmin**: xlim[1].
- **xmax**: xlim[2].
- **xcenter**: mean(xlim).
- **theta_lim**: Range of the angles on the spiral, measured in radians.
- **theta_min**: theta_lim[1].
- **theta_max**: theta_lim[2].
- **theta_center**: mean(theta_lim).
- **ylim**: Data range on y-axis.
- **ymin**: ylim[1].
- **ymax**: ylim[2].
- **ycenter**: mean(ylim).
- **rel_height**: Fraction of height of the track to the distance between two neighbouring loops.
- **abs_height**: The height of the track, which is rel_height multiplied by the distance between two neighbouring loops.
- **track_index**: Current track index.
Value

The corresponding value.

Examples

# There is no example
NULL
Index

$.TRACK_META, 38
arrow, 30
cartesian_to_xy, 2
current_spiral, 3
current_spiral_vp, 4
current_track_index, 5
cutree, 9
dendextend, 16
get_track_data, 5
grid.newpage, 21
grid.segments, 30
grid.text, 32
horizon_legend, 6
is_in_track, 7
Legend, 6
n_tracks, 8
names.TRACK_META, 7
phylo_to_dendrogram, 8
polar_to_cartesian, 9
print.TRACK_META, 10
set_current_track, 10, 35, 38
solve_theta_from_spiral_length, 11, 11
spiral_arrow, 12
spiral_axis, 13, 34
spiral_bars, 14
spiral_clear, 15
spiral_dendrogram, 16
spiral_highlight, 17
spiral_highlight_by_sector, 18
spiral_horizon, 6, 19
spiral_info, 20
spiral_initialize, 20, 22, 23, 36, 37
spiral_initialize_by_gcoor, 22
spiral_initialize_by_time, 23
spiral_lines, 24
spiral_opt, 25
spiral_phylo, 26
spiral_points, 27
spiral_polygon, 27
spiral_raster, 28
spiral_rect, 29
spiral_segments, 12, 30
spiral_text, 31
spiral_track, 33
spiral_xaxis, 34
spiral_yaxis, 34
Subset.TRACK_META ($.TRACK_META), 38
TRACK_META, 6, 35, 35
uniroot, 11
unit, 12, 13, 17, 18, 21, 27, 29, 31, 33, 35
viewport, 21
xy_to_cartesian, 36
xy_to_polar, 37