Package 'spiralize'

April 4, 2026

Type Package

Title Visualize Data on Spirals

Version 1.1.1

Date 2026-04-03

Depends R (>= 4.0.0), grid

Imports GlobalOptions (>= 0.1.1), GetoptLong (>= 0.1.8), circlize, stats, methods, grDevices, lubridate, utils, ComplexHeatmap

Suggests knitr, rmarkdown, grImport, grImport2, jpeg, png, tiff, cranlogs, cowplot, dendextend, bezier, magick, ape

Description It visualizes data along an Archimedean spiral <https://en.wikipedia.org/wiki/Archimedean_spiral>, makes so-called spiral graph or spiral chart.
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.

VignetteBuilder knitr

URL https://github.com/jokergoo/spiralize, https://jokergoo.github.io/spiralize/

License MIT + file LICENSE

NeedsCompilation no

RoxygenNote 7.3.1

Encoding UTF-8

Author Zuguang Gu [aut, cre] (ORCID: <https://orcid.org/0000-0002-7395-8709>)

Maintainer Zuguang Gu <guzuguang@suat-sz.edu.cn>

Repository CRAN

Date/Publication 2026-04-04 05:10:03 UTC

Contents

`current_spiral` *Get the current spiral object*

Description

Get the current spiral object

Usage

```
current_spiral()
```

Details

The returned value is an object of `spiral` reference class. The following methods might be useful (assume the object is named `s`):

- `s$curve()`: It returns the radius for given angles (in radians).
- `s$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 `s$spiral_length(theta2) - s$spiral_length(theta1)`.

Also there are the following meta-data for the current spiral:

- `s$xlim`: Data range.
- `s$xrange`: `s$xlim[2] - s$xlim[1]`.
- `s$theta_lim`: The corresponding range of theta.
- `s$theta_range`: `s$theta_lim[2] - s$theta_lim[1]`.
- `s$spiral_length_lim`: The corresponding range of spiral length.
- `s$spiral_length_range`: `s$spiral_length_lim[2] - s$spiral_length_lim[1]`.
- `s$max_radius`: Radius at `s$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 the current spiral*

Description

Viewport name of the current spiral

Usage

```
current_spiral_vp()
```

Value

A string of the viewport name.

current_track_index *Helper functions for handling tracks*

Description

Helper functions for handling tracks

Usage

```
current_track_index()

set_current_track(track_index)

n_tracks()

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

Arguments

track_index	The index of the track.
x	X-location of data points.
y	Y-location of data points.

Details

`is_in_track()` tests whether data points are inside a certain track.

Value

`current_track_index()` returns the index of the current track.

`set_current_track()` returns no value.

`n_tracks()` returns the number of available tracks.

`is_in_track()` returns a logical vector.

get_track_data *Meta-data of a track*

Description

Meta-data of a track

Usage

```
get_track_data(field, track_index = current_track_index())
```

Arguments

field	Name of the field, see the **Details** section.
track_index	The index of the track.

Details

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

- ymin: Minimal value on the y-axis.
- ymax: Maximal value on the y-axis.
- 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 spiral loops.

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

Value

A numeric vector (of length one or two) for the corresponding field.

---

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 ComplexHeatmap::Legend().

Value

A ComplexHeatmap::Legend object.

Examples

```
# see examples in `spiral_horizon()`.
```

`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. Let's say the form is `l = f(theta)` where `f()` is the complex equation for calculating `l`, `solve_theta_from_spiral_length()` tries to find theta with a known `l`. It uses `stats::uniroot()` to search for the solutions.

Value

The theta value.

Examples

```
spiral_initialize()
s = current_spiral()
theta = pi*seq(2, 3, length = 10)
theta
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

```
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 `grid::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`	Graphical parameters.
`track_index`	Index of the track.

Value

No value is returned.

See Also

Note `spiral_segments()` also supports drawing line-based arrows.

Examples

```
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

```
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()
)

spiral_xaxis(...)
```

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 `grid::unit()` object.
`minor_ticks_length`	Length of the minor ticks. The value should be a `grid::unit()` object.
`ticks_gp`	Graphical parameters for the ticks.
`labels_gp`	Graphical parameters for the labels.
`track_index`	Index of the track.
`...`	All pass to `spiral_axis()`.

Value

No value is returned.

Examples

```
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

```
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

```
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.

spiral_dendrogram *Draw dendrogram*

Description

Draw dendrogram

Usage

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

Arguments

dend	A stats::dendrogram object.
gp	Graphical parameters of the dendrogram edges, mainly as a global setting.
track_index	Index of the track.

Details

Graphical parameters for individual edges can be set via the edgePar attribute on each node in the dendrogram, see stats::dendrogram for how to set edgePar.

The dendrogram edges can also be rendered by dendextend::color_branches().

Value

Height of the dendrogram.

Examples

```
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))
spiral_dendrogram(dend)


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_dendrogram(dend)
```

`spiral_git_commits` *Visualize git commits*

Description

Visualize git commits

Usage

```
spiral_git_commits(
  repo = ".",
  show_legend = TRUE,
  start = NULL,
  end = Sys.Date(),
  pt_range = c(2, 16),
  commits_range = c(1, ceiling(quantile(n[n > 0], 0.95))),
  type = c("points", "heatmap"),
 colors = c("#3288BD", "#99D594", "#E6F598", "#FFFFBF", "#FEE08B", "#FC8D59", "#D53E4F")
)
```

Arguments

`repo`	Path of the git repo. The value can be a single repo or a vector of repos.
`show_legend`	Whether to show the legend.
`start`	Start date. By default it is the first date of the commit. The value can be a string such as "2022-01-01" or a `base::Date` object.
`end`	End date. By default it is the current date. The value can be a string such as "2022-01-01" or a `base::Date` object.
`pt_range`	Range of the point sizes. The default is between 1 and the 90 percentile of daily commits.
`commits_range`	Range of the numbers of commits.
`type`	Type of the plot.
`colors`	If type is the heatmap, it controls the list of colors.

Examples

```
## Not run:
spiral_git_commits("~/project/development/ComplexHeatmap")
spiral_git_commits("~/project/development/ComplexHeatmap", type = "heatmap")

## End(Not run)
```

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 `grid::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 `grid::unit()` object.
gp	Graphical 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

```
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 `grid::unit()` object with length one or two.
`gp`	Graphical 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

```
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 *Draw horizon chart along the spiral*

Description

Draw horizon chart along the spiral

Usage

```
spiral_horizon(
  x,
  y,
  y_max = max(abs(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.
y_max	Maximal absolute value on y-axis.
n_slices	Number of slices.
slice_size	Size of the slices. The final number of sizes is `ceiling(max(abs(y))/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 an 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.

See Also

`horizon_legend()` for generating the legend.

Examples

```
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)
spiral_track()
spiral_horizon(df$Date, df$Mean, use_bar = TRUE)

# with legend
require(ComplexHeatmap)
spiral_initialize_by_time(xlim = range(df$Date), unit_on_axis = "months", period = "year",
    period_per_loop = 20, polar_lines_by = 360/20,
    vp_param = list(x = unit(0, "npc"), just = "left"))
spiral_track()
lt = spiral_horizon(df$Date, df$Mean, use_bar = TRUE)
lgd = horizon_legend(lt, title = "Temperature difference")
draw(lgd, x = unit(1, "npc") + unit(2, "mm"), just = "left")
```

---

spiral_info *Information of the current spiral*

Description

Information of the current spiral

Usage

```
spiral_info()
```

Details

It prints information of the current spiral.

Value

No value is returned.

Examples

```
spiral_initialize()
spiral_track(ylim = c(0, 1), height = 0.4)
spiral_track(ylim = c(-10, 10), height = 0.4)
spiral_info()
```

---

spiral_initialize *Initialize the spiral*

Description

Initialize the spiral

Usage

```
spiral_initialize(
  xlim = c(0, 1),
  start = 360,
  end = 360 * 5,
  scale_by = c("angle", "curve_length"),
  period = NULL,
  clockwise = FALSE,
  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.

period	Under "angle" mode, the number of loops can also be controlled by argument period which controls the length of data a spiral loop corresponds to. Note in this case, argument end is ignored and the value for end is internally recalculated.
clockwise	Whether the curve is in a closewise direction. If it is set to TRUE, argument flip and reverse are ignored.
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. The value can also be a vector that defines where to add polar lines.
polar_lines_gp	Graphics parameters for the polar lines.
padding	Padding of the plotting region. The value can be a grid::unit() of length of one to two.
newpage	Whether to apply grid::grid.newpage() before making the plot?
vp_param	A list of parameters sent to grid::viewport().

Value

No value is returned.

Examples

```
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()
spiral_initialize(scale_by = "curve_length"); 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 formated 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"),
  normalize_year = FALSE,
  period_per_loop = 1,
  polar_lines_by = NULL,
  verbose = TRUE,
  ...
)
```

Arguments

`xlim`	Range of the time. The value can be time object such as `base::Date`, `base::POSIXlt` or `base::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?
`normalize_year`	Whether to enforce one spiral loop to represent a complete year?
`period_per_loop`	How many periods to put in a spiral 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 `peroid` is set to "hours". When `period` is year and `unit_on_axis` is day, the proportion of sectors by polar lines corresponds to the proportion of month days in a year.
`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 spiral loops.

Value

No value is returned.

Examples

```
spiral_initialize_by_time(xlim = c("2014-01-01", "2021-06-17"))
spiral_track(height = 0.6)
spiral_axis()

spiral_initialize_by_time(xlim = c("2021-01-01 00:00:00", "2021-01-05 00:00:00"))
spiral_track(height = 0.6)
spiral_axis()

spiral_initialize_by_time(xlim = c("2021-01-01 00:00:00", "2021-01-01 00:10:00"),
    unit_on_axis = "secs", period = "mins")
spiral_track(height = 0.6)
spiral_axis()
```

---

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 colors of the areas.
track_index	Index of the track.

Value

No value is returned.

Examples

```
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))
```

---

spiral_opt *Global options*

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 the following global parameters:

- min_segment_len Minimal length of the segment that partitions a curve.
- help Whether to print the help messages?

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.

Examples

```
spiral_opt
```

`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()
)

phylo_to_dendrogram(obj, log = FALSE)
```

Arguments

`obj`	A `stats::dendrogram` object.
`gp`	Graphical parameters of the tree edges, mainly as a global setting.
`log`	Whether the height of the tree to be log-transformed `log10(x + 1)`?
`reverse`	Whether the tree to 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.

Details

`phylo_to_dendrogram()` converts a `phylo` object to a `dendrogram` object.

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 stats::plot.dendrogram() function. However, you can still apply stats::cutree() to the returned dendrogram object with no problem and the dendrogram can be properly drawn with the **ComplexHeatmap** package (see examples).

Value

Height of the phylogenetic tree.

A stats::dendrogram object.

Examples

```
if(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)
}
if(require(ape)) {
    data(bird.families)
    d = phylo_to_dendrogram(bird.families)

    ComplexHeatmap::grid.dendrogram(d, test = TRUE)
}
```

---

spiral_pkg_downloads *Visualize package downloads*

Description

Visualize package downloads

Usage

```
spiral_pkg_downloads(
  pkg,
  from = "2012-10-01",
  to = "last-day",
  show_legend = TRUE
)
```

Arguments

pkg	A single CRAN package name.
from	Starting date.
to	Ending date.
show_legend	Whether to show the legend.

Details

The **cranlogs** package is used to retrieve the download history from the Rstudio server.

Examples

```
spiral_pkg_downloads("ggplot2")
```

`spiral_points` *Add points to a track*

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 `grid::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 *Add polygons to a track*

Description

Add polygons to a track

Usage

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

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.

Value

No value is returned.

Examples

```
x = seq(0, 2*pi*10, length = 1000)
y = c(sin(x), cos(rev(x)))
x2 = c(x, rev(x))

# in the normal cartesian coordinate system
plot(NULL, xlim = range(x2), ylim = range(y))
polygon(x2, y, col = "red")

# in the spiral coordinate system
spiral_initialize(xlim = range(x2))
spiral_track(ylim = range(y))
spiral_polygon(x2, y, gp = gpar(fill = "red"))

# try a different scale
spiral_initialize(xlim = range(x2), scale_by = "curve_length")
spiral_track(ylim = range(y))
spiral_polygon(x2, y, gp = gpar(fill = "red"))
```

spiral_raster *Add image to a track*

Description

Add image to a track

Usage

```
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.

Details

When `facing` is set to one of `"downward"`, `"inside"` and `"outside"`, both of `width` and `height` should be `grid::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.

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()
)
```

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 grid::arrow() object.
track_index	Index of the track.

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()
col = circlize::rand_color(n, luminosity = "bright")
spiral_segments(x0, y0, x1, y1,
    arrow = arrow(length = unit(2, "mm")), gp = gpar(col = col))

# if the segments are short and you want the straight "real" segments
spiral_initialize(xlim = range(c(x0, x1)))
spiral_track()
df0 = xy_to_cartesian(x0, y0)
df1 = xy_to_cartesian(x1, y1)
grid.segments(df0$x, df0$y, df1$x, df1$y, default.units = "native",
    arrow = arrow(length = unit(2, "mm")), gp = gpar(col = col))
```

---

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 `grid::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::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.
...	Pass to `grid::grid.text()`.

Details

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

Value

No value is returned.

Examples

```
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 = "curved_outside")
```

`spiral_track` *Add a new track or move to an existed track*

Description

Add a new track or move to an existed track

Usage

```
spiral_track(
  ylim = c(0, 1),
  height = 0.8,
  background = TRUE,
  background_gp = gpar(fill = "#EEEEEE"),
  reverse_y = FALSE,
  gradient = 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 spiral loops. The value can also be a `grid::unit()` object.
`background`	Whether to draw the background of the track, i.e. border and filled color of background.
`background_gp`	Graphical parameters of the background.
`reverse_y`	Whether reverse the direction of y-axis (i.e. pointing to the center of the spiral)?
`gradient`	Whether draw the background in gradient? The value can be a positive integer of the number of gradients from `background_gp$fill` to white.
`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_initialize()
spiral_track(height = 0.8, gradient = TRUE) # by default 10 gradients

spiral_initialize()
spiral_track(height = 0.8, background_gp = gpar(fill = "red"), gradient = 5)
```

---

spiral_yaxis *Draw y-axis*

Description

Draw y-axis

Usage

```
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 in spiral_initialize(), then "start" corresponds to the outside of the spiral.
at	Break points.
labels	Corresponding labels for the break points.
ticks_length	Length of the tick. Value should be a grid::unit() object.

ticks_gp	Graphical parameters for ticks.
labels_gp	Graphical 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

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

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

## S3 method for class 'TRACK_META'
x[[i, exact = TRUE]]

## S3 method for class 'TRACK_META'
x[i]

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

Arguments

x	The TRACK_META object.
name	Name of the meta name. For all supported names, type names(TRACK_META).
i	Name of the meta name. For all supported names, type names(TRACK_META).
exact	Please ignore.
...	Additional parameters.

Format

An object of class TRACK_META of length 1.

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 change the current track.

Don't directly use TRACK_META. The value of TRACK_META itself is meaningless. Always use in form of TRACK_META$name.

There are the following meta data for the current track:

- xlim: Data range on x-axis.
- xmin: xlim[1].
- xmax: xlim[2].
- xrange: xlim[2] - xlim[1].
- 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_range: theta_lim[2] - theta_lim[1].
- theta_center: mean(theta_lim).
- ylim: Data range on y-axis.
- ymin: ylim[1].
- ymax: ylim[2].
- yrange: ylim[2] - ylim[1].
- 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

```
spiral_initialize(xlim = c(0, 1))
spiral_track(ylim = c(0, 1))
for(nm in names(TRACK_META)) {
    cat(nm, ":\n", sep = "")
    print(TRACK_META[[nm]])
    cat("\n")
}
names(TRACK_META)
```

xy_to_cartesian *Transform between coordinate systems*

Description

Transform between coordinate systems

Usage

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

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

polar_to_cartesian(theta, r)

cartesian_to_polar(x, y)

cartesian_to_xy(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.
flip	If it is FALSE, it returns theta for the original spiral (before flipping).
theta	Angles, in radians.
r	Radius.

Details

There are three coordinate systems: the data coordinate system (xy), the polar coordinate system (polar) and the canvas coordinate system (cartesian). The canvas coordinates correspond to the "native" coordinates of the viewport where the graphics are drawn.

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

xy_to_cartesian() converts from the data coordinate system to the canvas coordinate system.

xy_to_polar() converts from the data coordinate system to the polar coordinate system.

polar_to_cartesian() converts from the polar coordinate system to the canvas coordinate system.

cartesian_to_polar() converts from the canvas coordinate system to the polar coordinate system.

cartesian_to_xy() converts from the canvas coordinate system to the data coordinate system. The data points are assigned to the nearest inner spiral loops (if the point is located inside a certain spiral loop, the distance is zero).

Value

`xy_to_cartesian()` returns A data frame with two columns: `x` and `y`.

`xy_to_polar()` returns a data frame with two columns: `theta` (in radians) and `r` (the radius).

`polar_to_cartesian()` returns a data frame with two columns: `x` and `y`.

`cartesian_to_polar()` returns a data frame with two columns: `theta` (in radians) and `r` (the radius).

`cartesian_to_xy()` returns a data frame with two columns: `x` and `y`.

Examples

```
x = runif(2)
y = runif(2)
spiral_initialize(xlim = c(0, 1))
spiral_track(ylim = c(0, 1))
spiral_points(x, y)
xy_to_cartesian(x, y)
xy_to_polar(x, y)

x = runif(100, -4, 4)
y = runif(100, -4, 4)
spiral_initialize(xlim = c(0, 1))
spiral_track(ylim = c(0, 1))
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, gp = gpar(col = 2))
```

Index