Package 'ggsector'

October 31, 2024

Title Draw Sectors

Version 1.7.0

Description Some useful functions that can use 'grid' and 'ggplot2' to plot sectors and interact with 'Seurat' to plot gene expression percentages. Also, there are some examples of how to draw sectors in 'ComplexHeatmap'.

License Artistic-2.0

Encoding UTF-8

RoxygenNote 7.3.2

Suggests BiocManager, ComplexHeatmap, knitr, rmarkdown, reshape2, utils

Depends ggplot2, grid

Imports dplyr, magrittr, Matrix, prettydoc, rlang, Seurat, stats, tibble, tidyr

VignetteBuilder knitr

BugReports https://github.com/yanpd01/ggsector/issues

NeedsCompilation no

Author Pengdong Yan [aut, cre, cph] (<https://orcid.org/0000-0002-2425-7930>)

Maintainer Pengdong Yan <yanpd01@gmail.com>

Repository CRAN

Date/Publication 2024-10-31 06:50:02 UTC

Contents

draw_key_sector *draw_key_sector*

Description

draw_key_sector

Usage

```
draw_key_sector(data, params, size)
```

Arguments

data	A single row data frame containing the scaled aesthetics to display in this key
params	A list of additional parameters supplied to the geom.
size	Width and height of key in mm.

Value

ggplot legend

GeomSectorPanel *ggplot sector*

Description

Draw sector with ggplot2.

Usage

```
geom_sector(
  mapping = NULL,
  data = NULL,
  stat = "identity",
  position = "identity",
  ...,
  na.rm = FALSE,
  show.legend = NA,
  inherit.aes = TRUE,
  individual = FALSE,
  verbose = TRUE
)
```

Arguments

mapping	Set of aesthetic mappings created by aes(). If specified and inherit.aes = TRUE (the default), it is combined with the default mapping at the top level of the plot. You must supply mapping if there is no plot mapping.
data	The data to be displayed in this layer. There are three options: If NULL, the default, the data is inherited from the plot data as specified in the call to ggplot(). A data.frame, or other object, will override the plot data. All objects will be fortified to produce a data frame. See fortify() for which variables will be created. A function will be called with a single argument, the plot data. The return value must be a data.frame, and will be used as the layer data. A function can be created from a formula (e.g. ~ head(.x, 10)).
stat	The statistical transformation to use on the data for this layer. When using a geom_*() function to construct a layer, the stat argument can be used the override the default coupling between geoms and stats. The stat argument accepts the following: • A Stat ggproto subclass, for example StatCount. • A string naming the stat. To give the stat as a string, strip the function name of the stat_ prefix. For example, to use stat_count(), give the stat as "count". • For more information and other ways to specify the stat, see the layer stat documentation.
position	A position adjustment to use on the data for this layer. This can be used in various ways, including to prevent overplotting and improving the display. The position argument accepts the following: • The result of calling a position function, such as position_jitter(). This method allows for passing extra arguments to the position. • A string naming the position adjustment. To give the position as a string, strip the function name of the position_ prefix. For example, to use position_jitter(), give the position as "jitter". • For more information and other ways to specify the position, see the layer position documentation.
...	Other arguments passed on to layer()'s params argument. These arguments broadly fall into one of 4 categories below. Notably, further arguments to the position argument, or aesthetics that are required can *not* be passed through .... Unknown arguments that are not part of the 4 categories below are ignored. • Static aesthetics that are not mapped to a scale, but are at a fixed value and apply to the layer as a whole. For example, colour = "red" or linewidth = 3. The geom's documentation has an **Aesthetics** section that lists the available options. The 'required' aesthetics cannot be passed on to the params. Please note that while passing unmapped aesthetics as vectors is technically possible, the order and required length is not guaranteed to be parallel to the input data.

- When constructing a layer using a `stat_*()` function, the `...` argument can be used to pass on parameters to the geom part of the layer. An example of this is `stat_density(geom = "area", outline.type = "both")`. The geom's documentation lists which parameters it can accept.
- Inversely, when constructing a layer using a `geom_*()` function, the `...` argument can be used to pass on parameters to the `stat` part of the layer. An example of this is `geom_area(stat = "density", adjust = 0.5)`. The stat's documentation lists which parameters it can accept.
- The `key_glyph` argument of `layer()` may also be passed on through `....` This can be one of the functions described as key glyphs, to change the display of the layer in the legend.

`na.rm` If `FALSE`, the default, missing values are removed with a warning. If `TRUE`, missing values are silently removed.

`show.legend` logical. Should this layer be included in the legends? `NA`, the default, includes if any aesthetics are mapped. `FALSE` never includes, and `TRUE` always includes. It can also be a named logical vector to finely select the aesthetics to display.

`inherit.aes` If `FALSE`, overrides the default aesthetics, rather than combining with them. This is most useful for helper functions that define both data and aesthetics and shouldn't inherit behaviour from the default plot specification, e.g. `borders()`.

`individual` Logical, default is FALSE. When "individual=FALSE", draw very quickly with a vector form, when "individual=TRUE", draw individually at a slower speed. Anyway, for better presentation, please add coord_fixed().

`verbose` Logical, default is TRUE. Whether to display reminder information.

Details

When "individual=FALSE", draw very quickly with a vector form, when "individual=TRUE", draw individually at a slower speed.

The required parameters in mapping are "x", "y", "theta", and the additional modifiable parameters are "r", "start", "r_start", "type", "colour", "fill", "ratio", "size" for line size, "linetype".

When there is coord_fixed(), r = 0.5 means that the sector-shaped background circle just fills the entire cell

The `ratio` parameter is still an experimental parameter, if it is not necessary, please do not set it yourself. The `ratio` parameter only works when `individual = FALSE`. When `ratio` is null, it will be auto calculated.

For better display effect, please always add `coord_fixed()`.

For details, please check the `grid.sector()`.

For more details, please type `vignette("ggsector")`.

Value

ggplot object

Examples

```
## prepare data
library(ggsector)
library(reshape2)
df <- cor(mtcars)[1:3, 1:5] %>%
    abs() %>%
    melt(varnames = c("x", "y"))

###
## Note, for better display effect, please always add coord_fixed()
## Note, for better display effect, please always add coord_fixed()
## Note, for better display effect, please always add coord_fixed()

## theta
ggplot(df) +
    ## type = "percent", theta = 0-100
    geom_sector(
        aes(y, x, theta = value * 100),
        type = "percent",
        color = "blue",
        individual = TRUE
    ) +
    ## type = "degree", theta = 0-360
    geom_sector(
        aes(y, x, theta = value * 360),
        type = "degree",
        color = "red",
        alpha = 0.5,
        individual = TRUE
    ) +
    coord_fixed() +
    theme_bw() +
    theme(axis.title = element_blank())

## r
ggplot(df) +
    geom_sector(
        aes(y, x, theta = value * 100),
        r = rep(c(0.15, 0.3, 0.45), 5),
        fill = 2,
        individual = TRUE
    ) +
    coord_fixed() +
    theme_bw() +
    theme(axis.title = element_blank())

## start
ggplot(df) +
    geom_sector(
        aes(y, x, theta = value * 100),
        start = rep(c(60, 40, 20), 5),
        fill = 2,
```

```
        individual = TRUE
    ) +
    coord_fixed() +
    theme_bw() +
    theme(axis.title = element_blank())

## r_start
ggplot(df) +
    geom_sector(
        aes(y, x, theta = value * 100),
        r_start = rep(c(0.15, 0.25, 0.35), 5),
        fill = 2,
        individual = TRUE
    ) +
    coord_fixed() +
    theme_bw() +
    theme(axis.title = element_blank())


####################  individual ###################
##########  individual with coord_fixed() ##########

## `individual = TRUE` + coord_fixed()
# x = x, y = y
ggplot(rbind(
    cbind(df, t1 = 1),
    cbind(df[1:9, ], t1 = 2)
)) +
    facet_wrap(~t1, ncol = 2) +
    geom_sector(
        aes(x, y),
        theta = 75,
        fill = 2,
        r = 0.5,
        individual = TRUE
    ) +
    coord_fixed() +
    theme_bw() +
    theme(axis.title = element_blank())

# x = y, y =x
ggplot(rbind(
    cbind(df, t1 = 1),
    cbind(df[1:9, ], t1 = 2)
)) +
    facet_wrap(~t1, ncol = 2) +
    geom_sector(
        aes(y, x),
        theta = 75,
        fill = 2,
        r = 0.5,
        individual = TRUE
    ) +
```

```
    coord_fixed() +
    theme_bw() +
    theme(axis.title = element_blank())

## `individual = FALSE` + coord_fixed()
# x = x, y = y
ggplot(rbind(
    cbind(df, t1 = 1),
    cbind(df[1:9, ], t1 = 2)
)) +
    facet_wrap(~t1, ncol = 2) +
    geom_sector(
        aes(x, y),
        theta = 75,
        fill = 2,
        r = 0.5,
        individual = FALSE
    ) +
    coord_fixed() +
    theme_bw() +
    theme(axis.title = element_blank())

# x = y, y =x
ggplot(rbind(
    cbind(df, t1 = 1),
    cbind(df[1:9, ], t1 = 2)
)) +
    facet_wrap(~t1, ncol = 2) +
    geom_sector(
        aes(y, x),
        theta = 75,
        fill = 2,
        r = 0.5,
        individual = TRUE
    ) +
    coord_fixed() +
    theme_bw() +
    theme(axis.title = element_blank())

##########  individual without coord_fixed() ##########
## If you are in a special situation and cannot use coord_fixed(),
## then it is recommended that you use `individual = TRUE` and
## the `r` parameter to fine-tune.
## Also, to reduce the radius, you need to try it manually.

## `individual = TRUE` without coord_fixed()
# x = x, y = y
ggplot(rbind(
    cbind(df, t1 = 1),
    cbind(df[1:9, ], t1 = 2)
)) +
    facet_wrap(~t1, ncol = 2) +
    geom_sector(
```

```
        aes(x, y),
        theta = 75,
        fill = 2,
        r = 0.35, ## To reduce the radius, you need to try it manually
        individual = TRUE
    ) +
    theme_bw() +
    theme(axis.title = element_blank())

# x = y, y =x
ggplot(rbind(
    cbind(df, t1 = 1),
    cbind(df[1:9, ], t1 = 2)
)) +
    facet_wrap(~t1, ncol = 2) +
    geom_sector(
        aes(y, x),
        theta = 75,
        fill = 2,
        r = 0.25, ## To reduce the radius, you need to try it manually
        individual = TRUE
    ) +
    theme_bw() +
    theme(axis.title = element_blank())

## `individual = FALSE`
## If you really want to use `individual = FALSE` without coord_fixed(),
## you might try the experimental parameter `ratio'
## You need to manually adjust the `ratio` value
## to prevent sector deformation.
# x = x, y = y
ggplot(rbind(
    cbind(df, t1 = 1),
    cbind(df[1:9, ], t1 = 2)
)) +
    facet_wrap(~t1, ncol = 2) +
    geom_sector(
        aes(x, y),
        theta = 75,
        fill = 2,
        r = 0.5,
        ## You need to manually adjust the `ratio` value
        ## to prevent sector deformation.
        ratio = 1.6,
        individual = FALSE
    ) +
    theme_bw() +
    theme(axis.title = element_blank())

# x = y, y =x
ggplot(rbind(
    cbind(df, t1 = 1),
    cbind(df[1:9, ], t1 = 2)
```

```
)) +
    facet_wrap(~t1, ncol = 2) +
    geom_sector(
        aes(y, x),
        theta = 75,
        fill = 2,
        r = 0.5,
        ## You need to manually adjust the `ratio` value
        ## to prevent sector deformation.
        ratio = 1.6,
        individual = FALSE
    ) +
    # coord_fixed() +
    theme_bw() +
    theme(axis.title = element_blank())
```

---

sectorGrob *Draw sector with grid*

Description

sectorGrob() return a polygon grob. grid.sector() draw sector. For more details, please type vignette("ggsector").

Usage

```
sectorGrob(
  x = 0.5,
  y = 0.5,
  theta = 25,
  r = 0.5,
  start = 0,
  r_start = 0,
  type = "percent",
  ratio = 1,
  group,
  default.units = "npc",
  vp = viewport(height = unit(1, "snpc"), width = unit(1, "snpc")),
  gp = gpar(col = "black", fill = "transparent")
)

grid.sector(
  x = 0.5,
  y = 0.5,
  theta = 25,
  r = 0.5,
```

```
  start = 0,
  r_start = 0,
  type = "percent",
  ratio = 1,
  group,
  default.units = "npc",
  vp = viewport(height = unit(1, "snpc"), width = unit(1, "snpc")),
  gp = gpar(col = "black", fill = "transparent")
)
```

Arguments

x	Numeric, the x-axis coordinate of the sector center.
y	Numeric, the y-axis coordinate of the sector center.
theta	Numeric, the angle of the sector, if 'type = "percent"', the input is a percentage(0-100), if 'type = "degree"', the input is an angle(0-360).
r	Numeric, radius of the outer circle of the sector(0-0.5).
start	Numeric, starting angle of sector.
r_start	Numeric, radius of the inner circle of the sector(0-r).
type	"percent", "degree" or an integer (preferably greater than 50), represents the number of scattered points on the circle where the sector is drawn. When `type = "percent"`, the circumference of the circle where the sector is located is composed of 100 scattered points; when `type = "degree"`, the circumference of the circle where the sector is located is composed of 360 scattered points; when `type = 150`, the circumference of the circle where the sector is located is composed of 150 scattered points.
ratio	aspect ratio, expressed as `y / x`.
group	A numeric vector used to separate locations in x and y into multiple sectors. If missing, it will be automatically added as a number.
default.units	A string indicating the default units to use if `x`, `y`, `width`, or `height` are only given as numeric vectors.
vp	A Grid viewport object (or NULL).
gp	An object of class `"gpar"`, typically the output from a call to the function `gpar`. This is basically a list of graphical parameter settings.

Value

polygon grob

draw sector

Examples

```
## Draw basic grid

# sectorGrob with units of "cm" and type of "degree"
grid.newpage()
```

```
gp <- sectorGrob(
    x = unit(c(3, 9, 15), "cm"),
    y = unit(c(5, 9, 15), "cm"),
    theta = c(90, 180, 270),
    r = 1,
    start = c(180, 180, 270),
    r_start = c(0.6, 0.3, 0),
    type = "degree",
    group = factor(1:3, levels = c(2, 3, 1)),
    gp = gpar(fill = c("green", "red", "blue"))
)
grid.draw(gp)

# grid.sector with units of "npc" and type of "percent"
grid.newpage()
grid.sector(
    x = c(0.1, 0.5, 0.9),
    y = c(0.9, 0.6, 0.1),
    theta = c(25, 50, 90),
    r = .1,
    start = c(25, 50, 100),
    r_start = c(0.06, 0.03, 0),
    type = "percent",
    group = factor(1:3, levels = c(2, 3, 1)),
    gp = gpar(col = c("green", "red", "blue"), fill = 2:4),
    default.units = "npc"
)


## Draw sector with ComplexHeatmap

# prepare data
library(magrittr)
library(ComplexHeatmap)
t0 <- cor(mtcars) %>%
    set_colnames(paste("y_", colnames(.))) %>%
    set_rownames(paste("x_", rownames(.)))
mat <- abs(t0)
mat[1:5, 1:5]

# Realized by modifying the [grid::viewport()],
# the sector can be set with a fixed width and height
set.seed(1)
Heatmap(
    mat,
    name = "vp",
    rect_gp = gpar(type = "none"),
    cell_fun = function(j, i, x, y, width, height, fill) {
        grid.rect(
            x = x, y = y, width = width, height = height,
            gp = gpar(col = "grey", fill = NA)
        )
        grid.sector(
```

```
            theta = mat[i, j] * 100,
            r = 0.5,
            start = mat[i, j] * 100 * runif(1),
            r_start = mat[i, j] * 0.49 * runif(1),
            vp = viewport(x, y, width, height),
            gp = gpar(fill = fill, col = "transparent")
        )
    },
    width = unit(.7, "snpc"),
    height = unit(.7, "snpc")
)

# Realized in the form of coordinates + radius.
# The default viewport locks the horizontal and vertical axes
# so that the sector does not deform, which needs to be removed here.
# The radius 'r' is half the min(length, width).
set.seed(2)
Heatmap(
    mat,
    name = "xy + r",
    rect_gp = gpar(type = "none"),
    cell_fun = function(j, i, x, y, width, height, fill) {
        grid.rect(
            x = x, y = y, width = width, height = height,
            gp = gpar(col = "grey", fill = NA)
        )
        r <- as.numeric(min(width, height)) / 2
        grid.sector(
            x,
            y,
            theta = mat[i, j] * 100,
            r = r,
            start = mat[i, j] * 100 * runif(1),
            r_start = mat[i, j] * r * 0.9 * runif(1),
            vp = NULL,
            gp = gpar(fill = fill, col = "transparent")
        )
    },
    width = unit(.7, "snpc"),
    height = unit(.7, "snpc")
)

# layer full
# The input matrix needs to be extracted with pindex(mat, i, j)
set.seed(3)
Heatmap(
    mat,
    name = "layer",
    rect_gp = gpar(type = "none"),
    layer_fun = function(j, i, x, y, width, height, fill) {
        grid.rect(
            x = x, y = y, width = width, height = height,
            gp = gpar(col = "grey", fill = NA)
```

```
        )
        r <- as.numeric(min(width, height)) / 2
        grid.sector(
            x,
            y,
            theta = pindex(mat, i, j) * 100,
            r = r,
            start = pindex(mat, i, j) * 100 * runif(nrow(mat) * ncol(mat)),
            r_start = pindex(mat, i, j) * r * 0.9 * runif(nrow(mat) * ncol(mat)),
            vp = NULL,
            gp = gpar(fill = fill, col = "transparent")
        )
    },
    width = unit(.7, "snpc"),
    height = unit(.7, "snpc")
)
```

`SectorPlot` *Draw sector for seurat object*

Description

A better alternative to `Seurat::DotPlot()`. For more details, please type `vignette("ggsector")`.

Usage

```
SectorPlot(
  object,
  features,
  features.level,
  assay,
  slot = c("data", "scale.data", "counts"),
  group.by,
  group.level,
  split.by,
  split.level,
  col_low = "blue",
  col_mid = "white",
  col_high = "red",
  col_midpoint,
  ...
)
```

Arguments

`object`	Seurat object

features	Input vector of genes list.
features.level	Levels of genes list.
assay	Specific assay to get data from or set data for; defaults to the default assay.
slot	Specific assay data to get or set.
group.by	Column of metadata to group the cells by, default is Idents().
group.level	Levels of group.
split.by	Column of metadata to split the cells by, default is NULL.
split.level	Levels of split vars.
col_low	Colours for low ends of the gradient.
col_mid	Colour for mid point.
col_high	Colours for high ends of the gradient.
col_midpoint	The midpoint (in data value) of the diverging scale.
...	Other arguments for ggplot2::facet_wrap(). Defaults to quantile(exp, 0.5)

Value

ggplot

Examples

```
## Download pbmc data from
# https://cf.10xgenomics.com/samples/cell/pbmc3k/pbmc3k_filtered_gene_bc_matrices.tar.gz
library(Seurat)
path <- paste0(tempdir(), "/pbmc3k.tar.gz")
file <- paste0(tempdir(), "/filtered_gene_bc_matrices/hg19")
download.file(
   "https://cf.10xgenomics.com/samples/cell/pbmc3k/pbmc3k_filtered_gene_bc_matrices.tar.gz",
    path
)
untar(path, exdir = tempdir())
pbmc.data <- Read10X(data.dir = file)
pbmc <- CreateSeuratObject(
    counts = pbmc.data,
    project = "pbmc3k",
    min.cells = 3,
    min.features = 200
)
pbmc <- NormalizeData(pbmc)
pbmc <- FindVariableFeatures(pbmc, selection.method = "vst", nfeatures = 2000)
pbmc <- ScaleData(pbmc, features = rownames(pbmc))
pbmc <- RunPCA(pbmc)
pbmc <- RunUMAP(pbmc, dim = 1:10)
pbmc <- FindNeighbors(pbmc, dims = 1:10)
pbmc <- FindClusters(pbmc, resolution = 1)
pbmc <- FindClusters(pbmc, resolution = 0.5)
markers <- tibble::tribble(
    ~type, ~marker,
```

```
    "Naive CD4+ T", "IL7R,CCR7",
    "CD14+ Mono", "CD14,LYZ",
    "Memory CD4+", "IL7R,S100A4",
    "B", "MS4A1",
    "CD8+ T", "CD8A",
    "FCGR3A+ Mono", "FCGR3A,MS4A7",
    "NK", "GNLY,NKG7",
    "DC", "FCER1A,CST3",
    "Platelet", "PPBP",
) %>%
    tidyr::separate_rows(marker, sep = ", *") %>%
    dplyr::distinct()

# Dotplot
DotPlot(pbmc, features = unique(markers$marker)) + coord_flip()

# contrast with DotPlot
SectorPlot(pbmc, markers$marker, features.level = unique(rev(markers$marker)))

SectorPlot(pbmc, markers$marker, group.by = "RNA_snn_res.1")

# split plot
# Assume a variable 'day', expressed as the number of days of cell development.
set.seed(1)
pbmc[["day"]] <- sample(1:3, ncol(pbmc), TRUE)
SectorPlot(pbmc, markers$marker, group.by = "RNA_snn_res.0.5", split.by = "day")
SectorPlot(
    pbmc, markers$marker,
    group.by = "day", split.by = "RNA_snn_res.0.5", nrow = 1
)
```

`sector_df` *sector coordinates*

Description

According to the input center position, radius and angle, get the polygon coordinates of a sector.

Usage

```
sector_df(
  x = 0.5,
  y = 0.5,
  theta = 25,
  r = 0.5,
  start = 0,
  r_start = 0,
  type = "percent",
```

```
  ratio = 1
)

sector_df_multiple(
  x = 0.5,
  y = 0.5,
  theta = 25,
  r = 0.5,
  start = 0,
  r_start = 0,
  type = "percent",
  ratio = 1,
  group
)
```

Arguments

x	Numeric, the x-axis coordinate of the sector center.
y	Numeric, the y-axis coordinate of the sector center.
theta	Numeric, the angle of the sector, if 'type = "percent"', the input is a percentage(0-100), if 'type = "degree"', the input is an angle(0-360).
r	Numeric, radius of the outer circle of the sector(0-0.5).
start	Numeric, starting angle of sector.
r_start	Numeric, radius of the inner circle of the sector(0-r).
type	"percent", "degree" or an integer (preferably greater than 50), represents the number of scattered points on the circle where the sector is drawn. When `type = "percent"`, the circumference of the circle where the sector is located is composed of 100 scattered points; when `type = "degree"`, the circumference of the circle where the sector is located is composed of 360 scattered points; when `type = 150`, the circumference of the circle where the sector is located is composed of 150 scattered points.
ratio	aspect ratio, expressed as `y / x`.
group	A numeric vector used to separate locations in x and y into multiple sectors. If missing, it will be automatically added as a number.

Details

`sector_df()` Only one value can be passed in for each parameter, and a sector coordinate is returned.

`sector_df_multiple()` Each parameter can pass in multiple values, and return multiple sector coordinates

The value of the 'type' parameter is "percent", "degree" or an integer (preferably greater than 50), represents the number of scattered points on the circle where the sector is drawn. When type = "percent", the circumference of the circle where the sector is located is composed of 100 scattered points; when type = "degree", the circumference of the circle where the sector is located is composed of 360 scattered points

For more details, please type `vignette("ggsector")`.

Value

coordinates of sector.

coordinates of sectors.

Examples

```
## coordinates of single sector
# type of percent, start = 0, r_start = 0
tmp_df <- sector_df(x = 0.5, y = 0.5, theta = 25, r = 0.4, start = 0, r_start = 0)
tmp_df
grid.newpage()
grid.polygon(
    tmp_df$x, tmp_df$y,
    vp = viewport(height = unit(1, "snpc"), width = unit(1, "snpc"))
)
# type of percent, start = 50, r_start = 0.2
tmp_df <- sector_df(x = 0.5, y = 0.5, theta = 25, r = 0.4, start = 50, r_start = 0.2)
tmp_df
grid.newpage()
grid.polygon(
    tmp_df$x, tmp_df$y,
    vp = viewport(height = unit(1, "snpc"), width = unit(1, "snpc"))
)

# type of degree, start = 90, r_start = 0
tmp_df <- sector_df(
    x = 0.5, y = 0.5, theta = 180, r = 0.4,
    start = 90, r_start = 0, type = "degree"
)
tmp_df
grid.newpage()
grid.polygon(
    tmp_df$x, tmp_df$y,
    vp = viewport(height = unit(1, "snpc"), width = unit(1, "snpc"))
)
# type of degree, start = 180, r_start = 0.2
tmp_df <- sector_df(
    x = 0.5, y = 0.5, theta = 180, r = 0.4,
    start = 270, r_start = 0.2, type = "degree"
)
tmp_df
grid.newpage()
grid.polygon(
    tmp_df$x, tmp_df$y,
    vp = viewport(height = unit(1, "snpc"), width = unit(1, "snpc"))
)

## Coordinates of Multiple Sectors
tmp_df <- sector_df_multiple(
    x = c(0.2, 0.5, 0.8),
    theta = c(25, 50, 75),
    r = 0.15,
```

```
    start = c(75, 50, 100),
    r_start = c(0, 0.05, 0.1),
    type = "percent"
)
tmp_df
grid.newpage()
grid.polygon(
    tmp_df$x,
    tmp_df$y,
    id = tmp_df$group,
    vp = viewport(height = unit(1, "snpc"), width = unit(1, "snpc")),
    gp = gpar(
        fill = 3:1, col = 1:3
    )
)

# type = 10, 100, 1000
tmp_df <- sector_df_multiple(
    x = c(0.25, 0.5, 0.75),
    theta = c(7.5, 75, 750),
    r = 0.125,
    r_start = c(0.05),
    type = c(c(10, "percent", 1000))
)
tmp_df
grid.newpage()
grid.polygon(
    tmp_df$x,
    tmp_df$y,
    id = tmp_df$group,
    vp = viewport(height = unit(1, "snpc"), width = unit(1, "snpc")),
    gp = gpar(
        fill = 3:1, col = 1:3
    )
)
```

Index