Package 'HiveR'

May 7, 2026

Type Package

Title 2D and 3D Hive Plots for R

Version 0.4.0

Date 2024-07-17

Description Creates and plots 2D and 3D hive plots. Hive plots are a unique method of displaying networks of many types in which node properties are mapped to axes using meaningful properties rather than being arbitrarily positioned. The hive plot concept was invented by Martin Krzywinski at the Genome Science Center (www.hiveplot.net/). Keywords: networks, food webs, linnet, systems biology, bioinformatics.

License GPL-3

Imports grid, plyr, jpeg, png, RColorBrewer, utils, stats, rgl, tcltk, xtable

Suggests bipartite

URL https://github.com/bryanhanson/HiveR

ByteCompile TRUE

BugReports https://github.com/bryanhanson/HiveR/issues

Depends R (>= 3.0)

Encoding UTF-8

RoxygenNote 7.3.2

NeedsCompilation no

Author Bryan A. Hanson [aut, cre] (ORCID: <https://orcid.org/0000-0003-3536-8246>), Vesna Memisevic [ctb], Jonathan Chung [ctb]

Maintainer Bryan A. Hanson <hanson@depauw.edu>

Repository CRAN

Date/Publication 2024-07-18 10:30:05 UTC

Contents

`HiveR-package` *2D and 3D Hive Plots for R*

Description

Creates and plots 2D and 3D hive plots. Hive plots are a unique method of displaying networks of many types in which node properties are mapped to axes using meaningful properties rather than being arbitrarily positioned. The hive plot concept was invented by Martin Krzywinski at the Genome Science Center (www.hiveplot.net/). Keywords: networks, food webs, linnet, systems biology, bioinformatics.

Author(s)

Bryan A. Hanson, DePauw University, Greencastle Indiana USA

See Also

Useful links:

- `https://github.com/bryanhanson/HiveR`
- Report bugs at `https://github.com/bryanhanson/HiveR/issues`

adj2HPD *Process an Adjacency Graph into a HivePlotData Object*

Description

This function will take an adjacency graph and convert it into a basic HivePlotData object. Further manipulation by mineHPD will almost certainly be required before the data can be plotted.

Usage

```
adj2HPD(M = NULL, axis.cols = NULL, type = "2D", desc = NULL, ...)
```

Arguments

M	A matrix with named dimensions. The names should be the node names. Should not be symmetric. If it is, only the lower triangle is used and a message is given.
axis.cols	A character vector giving the colors desired for the axes.
type	One of c("2D", "3D"). If 2D, a HivePlotData object suitable for use with plotHive will be created and the eventual hive plot will be static and 2D. If 3D, the HivePlotData object will be suitable for a 3D interactive plot using plot3dHive.
desc	Character. A description of the data set.
...	Other parameters to be passed downstream.

Details

This function produces a "bare bones" HivePlotData object. The names of the dimensions of M are used as the node names. All nodes are given size 1, an id number (1:number of nodes), are colored black and are assigned to axis 1. The edges are all gray, and the weight is M[i,j]. The user will likely have to manually make some changes to the resulting HivePlotData object before plotting. Alternatively, mineHPD may be able to extract some information buried in the data, but even then, the user will probably need to make some adjustments. See the examples.

Value

A HivePlotData object.

Author(s)

Bryan A. Hanson, DePauw University. <hanson@depauw.edu> Vesna Memisevic contributed a fix that limited this function to bipartite networks (changed in v. 0.2-12).

See Also

dot2HPD and adj2HPD

Examples

```
### Example 1: a bipartite network
### Note: this first example has questionable scientific value!
### The purpose is to show how to troubleshoot and
### manipulate a HivePlotData object.

if (require("bipartite")) {
  data(Safariland, package = "bipartite") # This is a bipartite network

  # You may wish to do ?Safariland or ?Safari for background

  hive1 <- adj2HPD(Safariland, desc = "Safariland data set from bipartite")
  sumHPD(hive1)

  # Note that all nodes are one axis with radius 1. Process further:

  hive2 <- mineHPD(hive1, option = "rad <- tot.edge.count")
  sumHPD(hive2)

  # All nodes still on 1 axis but degree has been used to set radius

  # Process further:

  hive3 <- mineHPD(hive2, option = "axis <- source.man.sink")
  sumHPD(hive3, chk.all = TRUE)

  # Note that mineHPD is generating some warnings, telling us
  # that the first 9 nodes were not assigned to an axis.  Direct
  # inspection of the data shows that these nodes are insects
  # that did not visit any of the flowers in this particular study.

  # Pretty up a few things, then plot:

  hive3$edges$weight <- sqrt(hive3$edges$weight) * 0.5
  hive3$nodes$size <- 0.5
  plotHive(hive3)

  # This is a one-sided hive plot of 2 axes, which results
  # from the curvature of the splines.  We can manually fix
 # this by reversing the ends of edges as follows:

  for (n in seq(1, length(hive3$edges$id1), by = 2)) {
    a <- hive3$edges$id1[n]
    b <- hive3$edges$id2[n]
    hive3$edges$id1[n] <- b
    hive3$edges$id2[n] <- a
  }

  plotHive(hive3)

  ### Example 2, a simple random adjacency matrix
  set.seed(31)
```

```
  nr <- 20
  nc <- 15
  M <- matrix(floor(runif(nc * nr, 0, 10)), ncol = nc)
  colnames(M) <- sample(c(letters, LETTERS), nc, replace = FALSE)
  rownames(M) <- sample(c(letters, LETTERS), nr, replace = FALSE)
  hive4 <- adj2HPD(M)
  sumHPD(hive4)
}
```

`animateHive` *Animate One or More 3D Hive Plots with a Handy Controller*

Description

This function takes a list of `HivePlotData` objects of `type = "3D"` and plots each in its own `rgl` window using its own arguments, then adds a controller which handles rotation and scaling.

Usage

```
animateHive(hives = list(), cmds = list(), xy = 400, ...)
```

Arguments

`hives`	A list of `HivePlotData` objects.
`cmds`	A list of arguments corresponding to how you want each hive plotted.
`xy`	An integer giving the size of the `rgl` window in pixels.
`...`	Other parameters to be passed downstream to `rgl`.

Value

None. Side effect is one or more plots.

Warning

If you click the 'continue rotating' box on the controller window, be sure to unclick it and wait for the system to halt before closing any of the windows. If you close the controller w/o doing this, the remaining open windows with the hive plots will continue rotating endlessly and it seems you can't get their attention to close the windows.

Author(s)

Bryan A. Hanson, DePauw University. <hanson@depauw.edu>

Examples

```
## Not run:
require("rgl")
# Sillyness: let's draw different hives with different settings
# List of hives
t4 <- ranHiveData(type = "3D", nx = 4)
t5 <- ranHiveData(type = "3D", nx = 5)
t6 <- ranHiveData(type = "3D", nx = 6)
myhives <- list(t4, t5, t6)
# List of arguments to plot in different coordinate systems
cmd1 <- list(method = "abs", LA = TRUE, dr.nodes = FALSE, ch = 10)
cmd2 <- list(method = "rank", LA = TRUE, dr.nodes = FALSE, ch = 2)
cmd3 <- list(method = "norm", LA = TRUE, dr.nodes = FALSE, ch = 0.1)
mycmds <- list(cmd1, cmd2, cmd3)
#
animateHive(hives = myhives, cmds = mycmds)

## End(Not run)
```

`Arroyo` *Plant-Pollinator Data Sets in Hive Plot Data Format*

Description

Plant-pollinator data sets which were derived ultimately from Vasquez and Simberloff, 2003. These are two-trophic level systems that have almost exactly the same plants and pollinators. `Safari` is from an undisturbed area, while `Arroyo` is from a nearby location grazed by cattle. In the original publication, the data sets are called Safariland and Arroyo Goye. See Details for how the original data was converted.

Details

These data sets are `HivePlotData` objects. They were created from the datasets `Safariland` and `vazarr` in the package `bipartite`. The process was the same for each: 1. Plants were placed on one axis, pollinators on the other. 2. A radius was assigned by calculating d' using function `dfun` in package `bipartite`. d' is an index of specialization; higher values mean the plant or pollinator is more specialized. 3. Edge weights were assigned proportional to the square root of the normalized number of visits of a pollinator to a plant. Thus the width of the edge drawn is an indication of the visitation rate. 4. The number of visits were divided manually into 4 groups and used to assign edge colors ranging from white to red. The redder colors represent greater numbers of visits, and the color-coding is comparable for each data set.

Author(s)

Bryan A. Hanson, DePauw University, Greencastle Indiana USA

chkHPD *Verify the Integrity of a Hive Plot Data Object*

Description

This function inspects the classes of each part of a HPD as a means of verifying its integrity. A few other characteristics are checked as well.

Usage

```
chkHPD(HPD, confirm = FALSE)
```

Arguments

HPD	An object of S3 class HivePlotData.
confirm	Logical; if TRUE then a favorable result is affirmed in the console (problems are always reported).

Value

A logical value; TRUE is there is a problem, otherwise FALSE.

Author(s)

Bryan A. Hanson, DePauw University. <hanson@depauw.edu>

See Also

sumHPD which allows inspection (checking) of many properties of your HPD.

Examples

```
test4 <- ranHiveData(nx = 4)
good <- chkHPD(test4, confirm = TRUE)
# mess it up and do again
# next test is not run as it halts execution
## Not run:
test4$nodes$color <- as.factor(test4$nodes$color)
bad <- chkHPD(test4)

## End(Not run)
```

dot2HPD *Process a .dot Graph File into a Hive Plot Data Object*

Description

This function will read a .dot file containing a graph specification in the DOT language, and (optionally) using two other files, convert the information into a HivePlotData object.

Usage

```
dot2HPD(
  file = NULL,
  node.inst = NULL,
  edge.inst = NULL,
  axis.cols = NULL,
  type = "2D",
  desc = NULL,
  ...
)
```

Arguments

file	The path to the .dot file to be processed.
node.inst	The path to a .csv file containing instructions about how to map node tags in the .dot file to parameters in the HivePlotData object. May be NULL.
edge.inst	The path to a .csv file containing instructions about how to map edge tags in the .dot file to parameters in the HivePlotData object. May be NULL.
axis.cols	A character vector giving the colors desired for the axes.
type	One of c("2D", "3D"). If 2D, a HivePlotData object suitable for use with plotHive will be created and the eventual hive plot will be static and 2D. If 3D, the HivePlotData object will be suitable for a 3D interactive plot using plot3dHive.
desc	Character. A description of the data set.
...	Other parameters to be passed downstream.

Details

This function is currently agnostic with respect to whether or not the .dot graph is directed or not. Either type will be processed, but if the graph is directed, this will only be indirectly stored in the HivePlotData object (in that the first node of an edge in the .dot file will be in HPD$nodes$id1 and the second node of an edge will be in HPD$nodes$id2. This fact can be used; see the vignette and mineHPD. Keep in mind the .dot standard is fairly loose. This function has been tested to work with several .dot files, include those with multiple tag=value attributes (in such cases, a typical line in the dot file should be formatted like this: node_name [tag1 = value1, tag2 = value2];). If you have trouble, please file a issue at Github so I can track it down.

Value

A HivePlotData object.

Author(s)

Bryan A. Hanson, DePauw University. <hanson@depauw.edu>

See Also

See the vignette for an example of using this function. Use browseVignettes("HiveR") to produce the vignette.

adj2HPD for a means of importing adjacency matrices.

---

drawHiveSpline *Draw a 3D Spline as Part of a 3D Hive Plot*

Description

This function analyzes the edges of a HivePlotData object in order to draw 3D splines representing those edges. Each pair of nodes at the ends of an edge is identified, and a control point is computed. This information is passed to rcsr to work out the details.

Usage

```
drawHiveSpline(HPD, L_A = FALSE, ...)
```

Arguments

HPD	An object of S3 class HivePlotData.
L_A	Logical: should splines be drawn with line_antialias = TRUE?
...	Parameters to be passed downstream.

Value

None. A spline is added to the 3D hive plot in progress.

Author(s)

Bryan A. Hanson, DePauw University. <hanson@depauw.edu>

See Also

plot3dHive which calls this function and is the user interface.

edge2HPD *Process an Edge List into a Hive Plot Data Object*

Description

This function will take an edge list and convert it into a basic HivePlotData object. Further manipulation by mineHPD will almost certainly be required before the data can be plotted.

Usage

```
edge2HPD(edge_df = NULL, axis.cols = NULL, type = "2D", desc = NULL, ...)
```

Arguments

edge_df	A data frame containing edge list information. Columns should be node1, node2, edge weight (column names are arbitrary). Edge weight information is optional. If missing, edge weights will be set to 1.
axis.cols	A character vector giving the colors desired for the axes.
type	One of c("2D", "3D"). If 2D, a HivePlotData object suitable for use with plotHive will be created and the eventual hive plot will be static and 2D. If 3D, the HivePlotData object will be suitable for a 3D interactive plot using plot3dHive.
desc	Character. A description of the data set.
...	Other parameters to be passed downstream.

Details

This function produces a "bare bones" HivePlotData object. The user will likely have to make some changes manually to the resulting HivePlotData object before plotting. Alternatively, mineHPD may be able to extract some information buried in the data, but even then, the user might need to make some adjustments. See the examples.

Value

A HivePlotData object.

Author(s)

Jonathan H. Chung, with minor changes for consistency by Bryan A. Hanson.

See Also

dot2HPD and adj2HPD

Examples

```
# Create a simple edge list & process it
edges <- data.frame(
  lab1 = LETTERS[c(1:8, 7)],
  lab2 = LETTERS[c(2:4, 1:3, 4, 2, 2)],
  weight = c(1, 1, 2, 2, 3, 1, 2, 3, 1)
)

td <- edge2HPD(edge_df = edges, desc = "Test of edge2HPD")
td.out <- sumHPD(td, plot.list = TRUE)
# compare:
edges
td.out[, c(3, 7, 8)]
```

HEC *A HivePlotData Object of the Hair Eye Color Data Set*

Description

This is an HPD (HivePlotData object) derived from the built-in hair eye color data set (see ?HairEyeColor). It serves as a test 2D data set, and the example below shows how it was built. While every data set is different and will require a different approach, the example illustrates the general approach to building a hive plot from scratch, step-by-step.

Format

The format is described in detail at HPD.

Examples

```
# An example of building an HPD from scratch

### Step 0.  Get to know your data.

data(HairEyeColor) # see ?HairEyeColor for background
df <- data.frame(HairEyeColor) # str(df) is useful

# Frequencies of the colors can be found with:
eyeF <- aggregate(Freq ~ Eye, data = df, FUN = "sum")
hairF <- aggregate(Freq ~ Hair, data = df, FUN = "sum")
es <- eyeF$Freq / eyeF$Freq[4] # node sizes for eye
hs <- hairF$Freq / hairF$Freq[3] # node sizes for hair

### Step 1. Assemble a data frame of the nodes.

# There are 32 rows in the data frame, but we are going to
# separate the hair color from the eye color and thus
# double the number of rows in the node data frame
```

```
nodes <- data.frame(
  id = 1:64,
  lab = paste(rep(c("hair", "eye"), each = 32), 1:64, sep = "_"),
  axis = rep(1:2, each = 32),
  radius = rep(NA, 64)
)

for (n in 1:32) {
  # assign node radius based most common colors
  if (df$Hair[n] == "Black") nodes$radius[n] <- 2
  if (df$Hair[n] == "Brown") nodes$radius[n] <- 4
  if (df$Hair[n] == "Red") nodes$radius[n] <- 1
  if (df$Hair[n] == "Blond") nodes$radius[n] <- 3

  if (df$Eye[n] == "Brown") nodes$radius[n + 32] <- 1
  if (df$Eye[n] == "Blue") nodes$radius[n + 32] <- 2
  if (df$Eye[n] == "Hazel") nodes$radius[n + 32] <- 3
  if (df$Eye[n] == "Green") nodes$radius[n + 32] <- 4

  # now do node sizes
  if (df$Hair[n] == "Black") nodes$size[n] <- hs[1]
  if (df$Hair[n] == "Brown") nodes$size[n] <- hs[2]
  if (df$Hair[n] == "Red") nodes$size[n] <- hs[3]
  if (df$Hair[n] == "Blond") nodes$size[n] <- hs[4]

  if (df$Eye[n] == "Brown") nodes$size[n + 32] <- es[4]
  if (df$Eye[n] == "Blue") nodes$size[n + 32] <- es[3]
  if (df$Eye[n] == "Hazel") nodes$size[n + 32] <- es[2]
  if (df$Eye[n] == "Green") nodes$size[n + 32] <- es[1]
}

nodes$color <- rep("black", 64)
nodes$lab <- as.character(nodes$lab) # clean up some data types
nodes$radius <- as.numeric(nodes$radius)

### Step 2. Assemble a data frame of the edges.

edges <- data.frame( # There will be 32 edges, corresponding to the original 32 rows
  id1 = c(1:16, 49:64), # This will set up edges between each eye/hair pair
  id2 = c(33:48, 17:32), # & put the males above and the females below
  weight = df$Freq,
  color = rep(c("lightblue", "pink"), each = 16)
)

edges$color <- as.character(edges$color)

# Scale the edge weight (det'd by trial & error to emphasize differences)
edges$weight <- 0.25 * log(edges$weight)^2.25

### Step 3. Now assemble the HivePlotData (HPD) object.

HEC <- list()
HEC$nodes <- nodes
```

```
HEC$edges <- edges
HEC$type <- "2D"
HEC$desc <- "HairEyeColor data set"
HEC$axis.cols <- c("grey", "grey")
class(HEC) <- "HivePlotData"

### Step 4. Check it & summarize

chkHPD(HEC) # answer of FALSE means there are no problems
sumHPD(HEC)

### Step 5.  Plot it.

# A minimal plot
plotHive(HEC, ch = 0.1, bkgnd = "white")
# See ?plotHive for fancier options
```

HidingAnAxis *How to Hide An Axis in a Hive Plot, with Bonus 2 Plots on One Page*

Description

From time-to-time is useful to compare several hive plots based on related data (and you might wish to plot them side-by-side to facilitate comparison). Depending the nature of the data set and how it changes under the experimental design, some data sets may not have any nodes on a particular axis (and therefore, they don't participate in edges either). Let's say your system fundamentally has three axes, but in some data sets one of the axes has no nodes. When you plot them side-by-side, for visual comparison it is nice if all the plots, including the one with an empty axis, have the same general orientation. In other words, even if the data only requires two axes, you might want it plotted as if it had three axes for consistency in overall appearance.

Details

When an axis is present but doesn't have a node on it, this makes `plotHive` unhappy, but there is a simple solution. You simply put a dummy or phantom node on the empty axis. This is illustrated in the example below. Also demonstrated is a simple `grid`-based function for putting more than one plot on a device.

Author(s)

Bryan A. Hanson, DePauw University. <hanson@depauw.edu>

Examples

```
require("grid")

# Adjacency matrix describing the connectivity in 2-butanone
# H's on a single carbon collapsed into a group.
# Matrix entry is bond order.  CH3 is coded so the
```

```
# bond order between C & H is 3 (3 single C-H bonds)

dnames <- c("C1", "C2", "C3", "C4", "O", "HC1", "HC3", "HC4")

#                    C1, C2, C3, C4, O, HC1, HC3, HC4
butanone <- matrix(c(
  0, 1, 0, 0, 0, 3, 0, 0, # C1
  1, 0, 1, 0, 2, 0, 0, 0, # C2
  0, 1, 0, 1, 0, 0, 2, 0, # C3
  0, 0, 1, 0, 0, 0, 0, 3, # C4
  0, 2, 0, 0, 0, 0, 0, 0, # O
  3, 0, 0, 0, 0, 0, 0, 0, # HC1
  0, 0, 2, 0, 0, 0, 0, 0, # HC3
  0, 0, 0, 3, 0, 0, 0, 0
), # HC4
ncol = 8, byrow = TRUE,
dimnames = list(dnames, dnames)
)

butanoneHPD <- adj2HPD(
  M = butanone, axis.col = c("black", "gray", "red"),
  desc = "2-butanone"
)

# Fix up the nodes manually (carbon is on axis 1)
butanoneHPD$nodes$axis[5] <- 3L # oxygen on axis 3
butanoneHPD$nodes$axis[6:8] <- 2L # hydrogen on axis 2
butanoneHPD$nodes$color[5] <- "red"
butanoneHPD$nodes$color[6:8] <- "gray"

# Exaggerate the edge weights, which are proportional to the number of bonds
butanoneHPD$edges$weight <- butanoneHPD$edges$weight^2
butanoneHPD$edges$color <- rep("wheat3", 7)

plotHive(butanoneHPD,
  method = "rank", bkgnd = "white",
  axLabs = c("carbon", "hydrogen", "oxygen"),
  axLab.pos = c(1, 1, 1), axLab.gpar =
    gpar(col = c("black", "gray", "red"))
)

# Now repeat the process for butane

dnames <- c("C1", "C2", "C3", "C4", "HC1", "HC2", "HC3", "HC4")

#                   C1, C2, C3, C4, HC1, HC2, HC3, HC4
butane <- matrix(c(
  0, 1, 0, 0, 3, 0, 0, 0, # C1
  1, 0, 1, 0, 0, 2, 0, 0, # C2
  0, 1, 0, 1, 0, 0, 2, 0, # C3
  0, 0, 1, 0, 0, 0, 0, 3, # C4
  3, 0, 0, 0, 0, 0, 0, 0, # HC1
  0, 2, 0, 0, 0, 0, 0, 0, # HC2
```

```
  0, 0, 2, 0, 0, 0, 0, 0, # HC3
  0, 0, 0, 3, 0, 0, 0, 0
), # HC4
ncol = 8, byrow = TRUE,
dimnames = list(dnames, dnames)
)

butaneHPD <- adj2HPD(
  M = butane, axis.col = c("black", "gray"),
  desc = "butane"
)
butaneHPD$nodes$axis[5:8] <- 2L # hydrogen on axis 2
butaneHPD$nodes$color[5:8] <- "gray"
butaneHPD$edges$weight <- butaneHPD$edges$weight^2
butaneHPD$edges$color <- rep("wheat3", 7)

plotHive(butaneHPD,
  method = "rank", bkgnd = "white",
  axLabs = c("carbon", "hydrogen"),
  axLab.pos = c(1, 1), axLab.gpar = gpar(col = c("black", "gray"))
)

# butaneHPD has 2 axes.  If we wanted to compare to butanoneHPD effectively
# we should add a third dummy axis where the oxygen axis was in butanone
# You might want to look at str(butaneHPD) before beginning

dummy <- c(9, "dummy", 3, 1.0, 1.0, "white") # mixed data types
# but coerced to character
butaneHPD$nodes <- rbind(butaneHPD$nodes, dummy)
str(butaneHPD$nodes) # The data types are mangled from the rbind!

# Now coerce the data types to the standard of the class, and check it
butaneHPD$nodes$id <- as.integer(butaneHPD$nodes$id)
butaneHPD$nodes$axis <- as.integer(butaneHPD$nodes$axis)
butaneHPD$nodes$radius <- as.numeric(butaneHPD$nodes$radius)
butaneHPD$nodes$size <- as.numeric(butaneHPD$nodes$size)
str(butaneHPD$nodes)

chkHPD(butaneHPD) # OK! (False means there were no problems)
sumHPD(butaneHPD)

# Plot it

plotHive(butaneHPD,
  method = "rank", bkgnd = "white",
  axLabs = c("carbon", "hydrogen", "oxygen"),
  axLab.pos = c(1, 1, 1), axLab.gpar =
    gpar(col = c("black", "gray", "red"))
)

# Put 2 plots side-by-side using a little helper function

vplayout <- function(x, y) viewport(layout.pos.row = x, layout.pos.col = y)
```

```
# pdf("Demo.pdf", width = 10, height = 5) # Aspect ratio better
# default screen device

grid.newpage()
pushViewport(viewport(layout = grid.layout(1, 2)))
pushViewport(vplayout(1, 1)) # left plot

plotHive(butanoneHPD,
  method = "rank", bkgnd = "white",
  axLabs = c("carbon", "hydrogen", "oxygen"),
  axLab.pos = c(1, 1, 1), axLab.gpar =
    gpar(col = c("black", "gray", "red")), np = FALSE
)
grid.text("butanone",
  x = 0.5, y = 0.1, default.units = "npc",
  gp = gpar(fontsize = 14, col = "black")
)

popViewport(2)
pushViewport(vplayout(1, 2)) # right plot
grid.text("test2")

plotHive(butaneHPD,
  method = "rank", bkgnd = "white",
  axLabs = c("carbon", "hydrogen", "oxygen"),
  axLab.pos = c(1, 1, 1), axLab.gpar =
    gpar(col = c("black", "gray", "red")), np = FALSE
)
grid.text("butane",
  x = 0.5, y = 0.1, default.units = "npc",
  gp = gpar(fontsize = 14, col = "black")
)

# dev.off()
```

`HivePlotData` *Hive Plot Data Objects*

Description

In package `HiveR`, hive plot data sets are stored as an S3 class called `HivePlotData`, detailed below.

Structure

The structure of a `HivePlotData` object is a list of 6 elements, some of which are data frames, and an attribute, as follows:

element	*(element)*	*type*	*description*
$nodes		data frame	Data frame of node properties

	$id	int	Node identifier
	$lab	chr	Node label
	$axis	int	Axis to which node is assigned
	$radius	num	Radius (position) of node along the axis
	$size	num	Node size in pixels
	$color	chr	Node color
$edges		data frame	Data frame of edge properties
	$id1	int	Starting node id
	$id2	int	Ending node id
	$weight	num	Width of edge in pixels
	$color	chr	Edge color
$type		chr	Type of hive. See Note.
$desc		chr	Description of data
$axis.cols		chr	Colors for axes
- attr		chr "HivePlotData"	The S3 class designation.

Note

While `$edges$id1` and `$edges$id2` are defined as the starting and ending nodes of a particular edge, hive plots as currently implemented are not directed graphs (agnostic might be a better word).

`HPD$type` indicates the type of hive data: If `2D`, then the data is intended to be plotted with `hivePlot` which is a 2D plot with axes radially oriented, and (hopefully) no edges that cross axes. If `3D`, then the data is intended to be plotted with `plot3dHive` which gives an interactive 3D plot, with axes oriented in 3D.

Author(s)

Bryan A. Hanson, DePauw University. <hanson@depauw.edu>

See Also

`sumHPD` to summarize a `HivePlotData` object.
`chkHPD` to verify the integrity of a `HivePlotData` object.
`ranHiveData` to generate random `HivePlotData` objects for testing and demonstration.

Examples

```
test4 <- ranHiveData(nx = 4)
str(test4)
sumHPD(test4)
plotHive(test4)
```

manipAxis *Modify the Display of Axes and Nodes in a Hive Plot*

Description

This function modifies various aspects of a HivePlotData object. A typical use is to convert the radii from the native/absolute values in the original object to either a normalized value (0...1) or to a ranked value. The order of nodes on an axis can also be inverted, and an axis can be pruned (removed) from the HivePlotData object.

Usage

```
manipAxis(HPD, method, action = NULL, ...)
```

Arguments

HPD	An object of S3 class HivePlotData.
method	One of c("rank", "norm", "scale", "invert", "ranknorm","prune", "offset", "stretch") giving the type of modification to be made.
action	For method = c("scale", "invert", "offset", "stretch"), a numeric vector of the same length as the number of axes.
...	Arguments to be passed downstream. Needed in this case for when plotHive has arguments for grid that get laundered through manipAxis

Details

The rank method uses ties.method = "first" so that each node gets a unique radius. For pruning, the nodes and edges are removed and then the remaining axes are renumbered to start from one. Exercise caution!

For "scale" node radii will be multiplied by the corresponding value in this argument. For "invert" a value of -1 will cause the corresponding axis to be inverted. For "prune", a single value specifying the axis to be pruned should be given. For "offset" the values in "action" will be subtracted from the node radii. For "stretch", node radii will first be offset so that the minimum value is zero, then multiplied by the values in "action" to stretch the axis. Depending upon the desired effect, one might use "stretch" followed by "offset" or perhaps other combinations.

Value

A modified HivePlotData object.

Author(s)

Bryan A. Hanson, DePauw University. <hanson@depauw.edu>

Examples

```
data(HEC)
# The first 3 examples take advantage of the argument '...'
# in plotHive, which passes action through to manipAxis on the fly.
# For this particular data, norm and absolute scaling appear the same.

plotHive(HEC, bkgnd = "white") # default is absolute positioning of nodes
plotHive(HEC, method = "rank", bkgnd = "white")
plotHive(HEC, method = "norm", bkgnd = "white")

# In these examples, we'll explicitly use manipAxis and then plot
# in a separate step.  This is because trying to plot on the fly in
# these cases will result in absolute scaling (which we do use here,
# but one might not want to be forced to do so).

HEC2 <- manipAxis(HEC, method = "invert", action = c(-1, 1))
plotHive(HEC2, bkgnd = "white")
HEC3 <- manipAxis(HEC, method = "stretch", action = c(2, 3))
plotHive(HEC3, bkgnd = "white")
HEC4 <- manipAxis(HEC, method = "offset", action = c(0, 1.5))
plotHive(HEC4, bkgnd = "white")
```

`mineHPD` *Examine (mine) a Hive Plot Data Object and Extract Information Contained Within It*

Description

A `HivePlotData` `object`, especially one created fresh using `dot2HPD`, generally contains a lot of hidden information about the network described. This function can extract this hidden information. This function has `options` which are quite specific as to what they do. The user can easily write new options and incorporate them. This function can be called multiple times using different options to gradually modify the `HivePlotData` object.

Usage

```
mineHPD(HPD, option = "rad <- tot.edge.count")
```

Arguments

`HPD`	A `HivePlotData` object.
`option`	A character string giving the option desired. See Details for current options.

Details

`option` = `"rad <- tot.edge.count"` This option looks through the `HivePlotData` object and determines how many edges start or end on each node (the "degree"). This value is then assigned to the radius for that node.

option = "axis <- source.man.sink" This option examines the nodes and corresponding edges in a HivePlotData object to determine if the node is a source, manager or sink. A source node only has outgoing edges. A sink node only has incoming edges. A manager has both. Hence, this option treats the HivePlotData object as if it were directed in that the first node of an edge in will be in HPD$nodes$id1 and the second node of an edge will be in HPD$nodes$id2. As a result, this option produces a hive plot with 3 axes (note: sources are on axis 1, sinks on axis 2, and managers on axis 3). This concept is similar to the idea of FuncMap but the internals are quite different. See also dot2HPD for some details about processing .dot files in an agnostic fashion.

option = "remove orphans" removes nodes that have degree zero (no incoming or outgoing edges).

option = "remove zero edge" removes edges with length zero. Such edges cause an error because the spline cannot be drawn. This option combines the next two options.

option = "remove self edge" removes edges that start and end on the same node.

option = "remove virtual edge" removes virtual edges which are edges which involve different nodes but the nodes happen to be on the the same axis at the same radius.

option = "remove edges same axis" removes edges which start and end on the same axis.

Value

A modified HivePlotData object.

Author(s)

Bryan A. Hanson, DePauw University. <hanson@depauw.edu>

See Also

See the vignette for an example of using this function. Use browseVignettes("HiveR") to produce the vignette.

---

plot3dHive *Create (Plot) a 2D or 3D Hive Plot*

Description

These functions plot a HivePlotData object in either 2D or 3D, depending upon which function is called.

Usage

```
plot3dHive(
  HPD,
  ch = 1,
  dr.nodes = TRUE,
  method = "abs",
  axLabs = NULL,
  axLab.pos = NULL,
```

```
  LA = FALSE,
  ...
)

plotHive(
  HPD,
  ch = 1,
  method = "abs",
  dr.nodes = TRUE,
  bkgnd = "black",
  axLabs = NULL,
  axLab.pos = NULL,
  axLab.gpar = NULL,
  anNodes = NULL,
  anNode.gpar = NULL,
  grInfo = NULL,
  arrow = NULL,
  np = TRUE,
  anCoord = "local",
  ...
)
```

Arguments

HPD	An object of S3 class HivePlotData.
ch	Numeric; the size of the central hole in the hive plot.
dr.nodes	Logical; if TRUE nodes will be drawn.
method	Character. Passed to manipAxis (see there for allowed values - the default given above plots using the native or absolute coordinates of the data).
axLabs	A vector of character strings for the axis labels.
axLab.pos	Numeric; An offset from the end of the axis for label placement. Either a single value or a vector of values. If a single value, all labels are offset the same amount. If a vector of values, there should be a value for each axis. This allows flexibility with long axis names. The units depend upon the method employed (see Details).
LA	(Applies to plot3dHive only) Logical: should splines be drawn with line_antialias = TRUE? See Details.
...	Additional parameters to be passed downstream.
bkgnd	Any valid color specification. Used for the background color for plotHive.
axLab.gpar	(Applies to plotHive only) A list of name - value pairs acceptable to gpar. These control the label and arrow displays. See the examples.
anNodes	(Applies to plotHive only) The path to a csv file containing information for labeling nodes. If present, a line segment will be drawn from the node to the specified text. The text is positioned near the end of the line segment. The columns in the csv file must be named as follows (description and use in parentheses): node.lab (node label from HPD$nodes$lab), node.text (the text to be drawn on

the plot), angle (polar coordinates: angle at which to draw the segment), radius (polar coordinates: radius at which to draw the text), offset (additional distance along the radius vector to offset text), hjust, vjust (horizontal and vertical justification; nominally in [0...1] but fractional and negative values also work). The first two values will be treated as type `character`, the others as `numeric`.

`anNode.gpar` (Applies to `plotHive` only) A list of name - value pairs acceptable to `gpar`. These control both the text used to annotate the nodes and the line segments connecting that text to the node. See the examples.

`grInfo` (Applies to `plotHive` only) The path to a csv file containing information for adding graphic decorations to the plot. If present, a line segment will be drawn from the node to the specified location and the graphic is positioned near the end the line segment. The columns in the csv file must be named as follows (description and use in parentheses): node.lab (node label from HPD$nodes$lab), angle (polar coordinates: angle at which to position the graphic), radius (polar coordinates: radius at which to position the graphic), offset (additional distance along radius vector to offset the graphic), width (the width of the graphic), path (a valid path to the graphics in jpg or png format). The path should include the extension is it is autodetected. Valid extensions are jpg, JPG, jpeg, JPEG, png, or PNG. All image files must be of the same type (all jpg, or all png).

`arrow` (Applies to `plotHive` only) A vector of 5 or 6 values: a character string to label the arrow, and 4 numeric values giving the angle of the arrow, the radius at which to start the arrow, the radius at which to end the arrow, and a value to offset the arrow label from the end of the arrow. A 5th numeric value (the 6th argument overall) can specify an offset in the y direction for the arrow useful when `nx = 2`. See the examples.

`np` (Applies to `plotHive` only) Logical; should a new device (page) be opened when drawing the hive plot? If you are making multiple plots within some sort of `grid` scheme then this should be set to `FALSE`.

`anCoord` (Applies to `plotHive` only) One of `c("local","global")`. Controls how the position of node labels and graphic decorations are specified. See Details.

Details

General. `plotHive` uses `grid` graphics to produce a 2D hive plot in a style similar to the original concept. For a 2D plot, axis number 1 is vertical except in the case of 2 axes in which case it is to the right. `plot3dHive` produces a 3D hive plot using `rgl` graphics. Functions from either package can be used to make additional modifications after the hive plot is drawn, either via the ... argument or by subsequent function calls. See the examples.

Units and Annotations. If you add node labels, arrows or graphic decorations, the units that you must specify are those intrinsic to the data itself, modified by your setting of `ch` and `method`. These generally cannot be known precisely ahead of time, so some experimentation will be necessary to polish the plots. For instance, if you have data with node radii that run from 4-23 then you have an idea of how to position your annotations if using `method = "abs"`. But the same data plotted with `method = "norm"` or `method = "rank"` will require that you move your annotation positions accordingly. In the first case no radius is larger than 23, but the maximum radius is 1 when the data is normed and when it is ranked, the maximum value will depend upon which axis has the most nodes on it, and the number of unique radii values.

Positioning Node Labels and Graphics. In addition to the nuances just above, there are two ways to specify the location of node labels and graphic decorations. Polar coordinates are used in both cases. If `annCoord = "local"` then the angle, radius and offset arguments are relative to the node to be annotated. An angle of 0 positions the label horizontally to the right of the node. Thus the label can be placed within a circular area around the node. If `annCoord = "global"` then the specifications are relative to dead center on the plot. These two methods give one lots of flexibility in lining up labels in different ways. See the examples.

Size of Graphics. The size of graphic decorations is controlled by the column 'width' in `grInfo`. The ultimate call to display the graphic is done with `as.raster`. Specifying only the width preserves the aspect ratio of the graphic. See `?as.raster` for further discussion.

Colors. For any of the `gpar` arguments, watch out: In grid graphics the default color for text and arrows is black, so if are using the default `bkgnd = "black"` in the hive plot be sure to specify `col = "white"` (or some other non-black color) for the labels and arrows or you won't see them.

Speed and 3D Hive Plots. For most work with `plot3dHive`, use `LA = FALSE` for speed of drawing. `LA = TRUE` is over 20 times slower, and is more appropriate for high quality hive plots. These are probably better made with `R CMD BATCH script.R` rather than interactive use.

Value

None. Side effect is a plot.

Functions

- `plot3dHive()`: Create a 3D Hive Plot
- `plotHive()`: Create a 2D Hive Plot

Author(s)

Bryan A. Hanson, DePauw University. <hanson@depauw.edu>

Examples

```
### 2D Hive Plots
require("grid")
# Generate some random data
test2 <- ranHiveData(nx = 2)
test3 <- ranHiveData(nx = 3)

# First the nx = 2 case.
# Note that gpar contains parameters that apply to both the
# axis labels and arrow. A 6th value in arrow offsets the arrow vertically:
plotHive(test2,
  ch = 5, axLabs = c("axis 1", "axis 2"), rot = c(-90, 90),
  axLab.pos = c(20, 20), axLab.gpar = gpar(col = "pink", fontsize = 14, lwd = 2),
  arrow = c("radius units", 0, 20, 60, 25, 40)
)

# Now nx = 3:
plotHive(test3) # default plot
```

```
# Add axis labels & options to nx = 3 example.  Note that rot is not part of gpar
plotHive(test3,
  ch = 5, axLabs = c("axis 1", "axis 2", "axis 3"),
  axLab.pos = c(10, 15, 15), rot = c(0, 30, -30),
  axLab.gpar = gpar(col = "orange", fontsize = 14)
)

# Call up a built-in data set to illustrate some plotting tricks
data(HEC)
require("grid") # for text additions outside of HiveR (grid.text)

plotHive(HEC,
  ch = 0.1, bkgnd = "white",
  axLabs = c("hair\ncolor", "eye\ncolor"),
  axLab.pos = c(1, 1),
  axLab.gpar = gpar(fontsize = 14)
)
grid.text("males", x = 0, y = 2.3, default.units = "native")
grid.text("females", x = 0, y = -2.3, default.units = "native")
grid.text("Pairing of Eye Color with Hair Color",
  x = 0, y = 4,
  default.units = "native", gp = gpar(fontsize = 18)
)

# Add node labels and graphic decorations
# The working directory has to include
# not only the grInfo and anNodes files but also the jpgs.
# So, we are going to move to such a directory and return you home afterwards.

currDir <- getwd()
setwd(system.file("extdata", "Misc", package = "HiveR"))
plotHive(HEC,
  ch = 0.1, bkgnd = "white",
  axLabs = c("hair\ncolor", "eye\ncolor"),
  axLab.pos = c(1, 1),
  axLab.gpar = gpar(fontsize = 14),
  anNodes = "HECnodes.txt",
  anNode.gpar = gpar(col = "black"),
  grInfo = "HECgraphics.txt",
  arrow = c("more\ncommon", 0.0, 2, 4, 1, -2)
)

grid.text("males", x = 0, y = 2.3, default.units = "native")
grid.text("females", x = 0, y = -2.3, default.units = "native")
grid.text("Pairing of Eye Color with Hair Color",
  x = 0, y = 3.75,
  default.units = "native", gp = gpar(fontsize = 18)
)
grid.text("A test of plotHive annotation options",
  x = 0, y = 3.25,
  default.units = "native", gp = gpar(fontsize = 12)
)
grid.text("Images from Wikipedia Commons",
```

```
  x = 0, y = -3.5,
  default.units = "native", gp = gpar(fontsize = 9)
)
setwd(currDir)

# Use the node label concept to create tick marks

currDir <- getwd()
setwd(system.file("extdata", "Misc", package = "HiveR"))
plotHive(HEC,
  ch = 0.1, bkgnd = "white",
  axLabs = c("hair\ncolor", "eye\ncolor"),
  axLab.pos = c(1, 1),
  axLab.gpar = gpar(fontsize = 14),
  anNodes = "HECticks.txt",
  anNode.gpar = gpar(col = "black"),
  arrow = c("more\ncommon", 0.0, 2, 4, 1, -2),
  dr.nodes = FALSE
)

grid.text("males", x = 0, y = 2.3, default.units = "native")
grid.text("females", x = 0, y = -2.3, default.units = "native")
grid.text("Pairing of Eye Color with Hair Color",
  x = 0, y = 3.75,
  default.units = "native", gp = gpar(fontsize = 18)
)
grid.text("Adding tick marks to the nodes",
  x = 0, y = 3.25,
  default.units = "native", gp = gpar(fontsize = 12)
)
setwd(currDir)


### 3D Hive Plots. The following must be run interactively.
## Not run:
require("rgl")
test4 <- ranHiveData(nx = 4, type = "3D")
plot3dHive(test4)

## End(Not run)
```

---

ranHiveData *Generate Random Hive Plot Data*

Description

This function generates random data sets which can be used to make a hive plot.

Usage

```
ranHiveData(
  type = "2D",
  nx = 4,
  nn = nx * 15,
  ne = nx * 15,
  rad = 1:100,
  ns = c(0.5, 1, 1.5),
  ew = 1:3,
  nc = brewer.pal(5, "Set1"),
  ec = brewer.pal(5, "Set1"),
  axis.cols = brewer.pal(nx, "Set1"),
  desc = NULL,
  allow.same = FALSE,
  verbose = FALSE
)
```

Arguments

type	The type of hive plot to be generated. One of c("2D","3D").
nx	An integer giving the number of axes to be created (2 =< nx =< 6).
nn	An integer giving the number of nodes to be created. This is an initial number which may be reduced during clean up. See Details.
ne	An integer giving the number of edges to be created. This is an initial number which may be reduced during clean up. See Details.
rad	Numeric; a range of values that will be used as node radius values (the position of the node along the axis).
ns	Numeric; a range of values that will be used as the node sizes.
ew	Numeric; a range of values that will be used as the edge weights.
nc	A vector of valid color names giving the node colors.
ec	A vector of valid color names giving the edge colors.
axis.cols	A vector of valid color names to be used to color the axes; length(axis.cols) must = nx.
desc	Character; a description of the data set.
allow.same	Logical; indicates if edges may begin and end on the same axis. Only applies to type = 2D.
verbose	Logical; If TRUE, the generation, processing and final result is reported to the console.

Details

For type = "2D", after the function creates an initial set of random nodes, these are randomly chosen and connected between adjacent axes, so that no edge crosses an axis.

For type = "3D", after the function creates an initial set of random nodes and edges, these are

cleaned up by removing the following cases (which the rest of HiveR is not intended to handle at this time): duplicated nodes, nodes that are not part of any edge, edges that begin and end on the same point, edges that begin and end on the same axis, and finally, for nx = 5 or 6, edges that begin and end on colinear axes. Most of these don't cause an error, but produce some ugly results.

For the arguments rad, ns, ew, nc and ec, the values given are sampled randomly (with replacement) and assigned to particular nodes or edges.

Value

An object of S3 class HivePlotData.

Warning

If you create a very small data set with few nodes, there may be no nodes assigned to some axes which will give an error when you try to plot the data. It's up to the user to check for this possibility (you can use sumHPD).

Author(s)

Bryan A. Hanson, DePauw University. <hanson@depauw.edu>

Examples

```
test4 <- ranHiveData(nx = 4)
str(test4)
sumHPD(test4)
```

---

rcsr *Compute the Details of a 3D Spline for a Hive Plot Edge*

Description

This is a wild bit of trigonometry! Three points in 3D space, two ends and an control point, are rotated into 2D space. Then a spline curve is computed. This is necessary because spline curves are only defined in R as 2D objects. The new collection of points, which is the complete spline curve and when drawn will be the edge of a hive plot, is rotated back into the original 3D space. rcsr stands for rotate, compute spline, rotate back.

Usage

```
rcsr(p0, cp, p1)
```

Arguments

p0	A triple representing one end of the final curve (x, y, z).
cp	A triple representing the control point used to compute the final curve (x, y, z).
p1	A triple representing the other end of the final curve (x, y, z).

Details

See the code for exactly how the function works. Based upon the process described at http://www.fundza.com/mel/axis_to_vector/index.html Timing tests show this function is fast and scales linearly (i.e. 10x more splines to draw takes 10x more time). Roughly 3 seconds were required to draw 1,000 spline curves in my testing.

Value

A 3 column matrix with the x, y and z coordinates to be plotted to create a hive plot edge.

Author(s)

Bryan A. Hanson, DePauw University. <hanson@depauw.edu>

Examples

```
# This is a lengthy example to prove it works.
# Read it and then copy the whole thing to a blank script.
# Parts of it require rgl and are interactive.
# So none of the below is run during package build/check.

### First, a helper function
## Not run:

drawUnitCoord <- function() {

  # Simple function to draw a unit 3D coordinate system

  # Draw a Coordinate System

  r <- c(0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1) # in polar coordinates
  theta <- c(0, 0, 0, 90, 0, 180, 0, 270, 0, 0, 0, 0) # start, end, start, end
  phi <- c(0, 90, 0, 90, 0, 90, 0, 90, 0, 0, 0, 180)
  cs <- data.frame(radius = r, theta, phi)
  ax.coord <- sph2cart(cs)

  segments3d(ax.coord, col = "gray", line_antialias = TRUE)
  points3d(
    x = 0, y = 0, z = 0, color = "black", size = 4,
    point_antialias = TRUE
  ) # plot origin

  # Label the axes

  r <- c(1.1, 1.1, 1.1, 1.1, 1.1, 1.1) # in polar coordinates
  theta <- c(0, 90, 180, 270, 0, 0)
  phi <- c(90, 90, 90, 90, 0, 180)
  l <- data.frame(radius = r, theta, phi)
  lab.coord <- sph2cart(l)
  text3d(lab.coord, texts = c("+x", "+y", "-x", "-y", "+z", "-z"))
}
```

```
###  Now, draw a reference coordinate system and demo the function in it.

drawUnitCoord()

### Draw a bounding box

box <- data.frame(
  x = c(1, -1, 1, 1, 1, 1, 1, 1, -1, -1, -1, 1, 1, 1, -1, 1, -1, -1, -1, -1, -1, -1, -1),
  y = c(1, 1, 1, 1, 1, -1, 1, -1, 1, 1, 1, 1, -1, -1, -1, -1, -1, -1, -1, -1, -1, 1, -1, 1),
  z = c(1, 1, 1, -1, 1, 1, -1, -1, -1, -1, 1, -1, 1, -1, 1, 1, -1, -1, -1, 1, 1, 1, -1, -1)
)

segments3d(box$x, box$y, box$z, line_antialias = TRUE, col = "red")

### Draw the midlines defining planes

mid <- data.frame(
  x = c(0, 0, 0, 0, 0, 0, 0, 0, 1, -1, -1, -1, -1, 1, 1, 1, 1, -1, -1, -1, -1, 1, 1, 1),
  y = c(-1, -1, -1, 1, 1, 1, 1, -1, 0, 0, 0, 0, 0, 0, 0, 0, -1, -1, -1, 1, 1, 1, 1, -1),
  z = c(-1, 1, 1, 1, 1, -1, -1, -1, -1, -1, -1, 1, 1, 1, 1, -1, 0, 0, 0, 0, 0, 0, 0, 0)
)

segments3d(mid$x, mid$y, mid$z, line_antialias = TRUE, col = "blue")

### Generate two random points

p <- runif(6, -1, 1)

# Special case where p1 is on z axis
# Uncomment line below to demo
# p[4:5] <- 0

p0 <- c(p[1], p[2], p[3])
p1 <- c(p[4], p[5], p[6])

### Draw the pts, label them, draw vectors to those pts from origin

segments3d(
  x = c(0, p[1], 0, p[4]),
  y = c(0, p[2], 0, p[5]),
  z = c(0, p[3], 0, p[6]),
  line_antialias = TRUE, col = "black", lwd = 3
)

points3d(
  x = c(p[1], p[4]),
  y = c(p[2], p[5]),
  z = c(p[3], p[6]),
  point_antialias = TRUE, col = "black", size = 8
)

text3d(
  x = c(p[1], p[4]),
```

```
  y = c(p[2], p[5]),
  z = c(p[3], p[6]),
  col = "black", texts = c("p0", "p1"), adj = c(1, 1)
)

### Locate control point
### Compute and draw net vector from origin thru cp
### Connect p0 and p1

s <- p0 + p1
segments3d(
  x = c(0, s[1]), y = c(0, s[2]), z = c(0, s[3]),
  line_antialias = TRUE, col = "grey", lwd = 3
)

segments3d(
  x = c(p[1], p[4]), # connect p0 & p1
  y = c(p[2], p[5]),
  z = c(p[3], p[6]),
  line_antialias = TRUE, col = "grey", lwd = 3
)

cp <- 0.6 * s # Now for the control point

points3d(
  x = cp[1], # Plot the control point
  y = cp[2],
  z = cp[3],
  point_antialias = TRUE, col = "black", size = 8
)

text3d(
  x = cp[1], # Label the control point
  y = cp[2],
  z = cp[3],
  texts = c("cp"), col = "black", adj = c(1, 1)
)

### Now ready to work on the spline curve

n2 <- rcsr(p0, cp, p1) # Compute the spline

lines3d(
  x = n2[, 1], y = n2[, 2], z = n2[, 3],
  line_antialias = TRUE, col = "blue", lwd = 3
)

### Ta-Da!!!!!

## End(Not run)
```

sph2cart *Convert Spherical to Cartesian Coordinates*

Description

This function converts spherical to Cartesian coordinates.

Usage

```
sph2cart(df)
```

Arguments

df	A data frame with columns named r, theta and phi with the radius and angles (in spherical coordinates) to be converted to Cartesian coordinates.

Value

A data frame with named columns containing the converted coordinates.

Note

Cobbled together from similar functions in other packages.

Author(s)

Bryan A. Hanson, DePauw University. <hanson@depauw.edu>

sumHPD *Summarize a Hive Plot Data Object and Optionally Run Some Checks*

Description

This function summarizes a HivePlotData object in a convenient form. Optionally, it can run some checks for certain conditions that may be of interest. It can also output a summary of edges to be drawn, either as a data frame or in a LaTeX ready form, or a data frame of orphaned nodes.

Usage

```
sumHPD(
  HPD,
  chk.all = FALSE,
  chk.sm.pt = FALSE,
  chk.ax.jump = FALSE,
  chk.sm.ax = FALSE,
  chk.orphan.node = FALSE,
```

```
  chk.virtual.edge = FALSE,
  plot.list = FALSE,
  tex = FALSE,
  orphan.list = FALSE
)
```

Arguments

HPD	An object of S3 class HivePlotData.
chk.all	Logical; should all the checks below be run? See Details.
chk.sm.pt	Logical; should the edges be checked to see if any of them start and end on the same axis with the same radius? See Details.
chk.ax.jump	Logical; should the edges be checked to see if any of them start and end on non-adjacent axes, e.g. axis 1 –> axis 3? See Details.
chk.sm.ax	Logical; should the edges be checked to see if any of them start and end on the same axis?
chk.orphan.node	Logical; should orphan nodes be identifed? Orphan nodes have degree 0 (no incoming or outgoing edges).
chk.virtual.edge	Logical; should the edges be checked to see if any of them start and end on different nodes which happen to be at the same radius on the same axis? See Details.
plot.list	Logical; should a data frame of edges to be drawn be returned?
tex	Logical; should the plot.list be formatted for LaTeX?
orphan.list	Logical; should a data frame of orphaned nodes be returned?

Details

Argument chk.sm.pt applies only to hive plots of type = 2D. It checks to see if any of the edges start and end at the same node id. These by definition exist at the same radius on the same axis, which causes an error in plotHive since you are trying to draw an edge of length zero (the actual error message is Error in calcCurveGrob(x,x$debug) : End points must not be identical. Some data sets may have such cases intrinsically or due to data entry error, or the condition may arise during processing. Either way, one needs to be able to detect such cases for removal or modification. This argument will tell you which nodes cause the problem.

Argument chk.virtual.edge applies only to hive plots of type = 2D and is similiar to chk.sm.pt above except that it checks for virtual edges. These are edges start and end on the same axis at the same radius but at different node id's (in other words, two nodes have the same radius on the same axis). This condition gives the same error as above. It is checked for separately as it arises via a different problem in the construction of the data.

Argument chk.ax.jump applies only to hive plots of type = 2D. It checks to see if any of the edges jump an axis, e.g. axis 1 –> axis 3. This argument will tell you which nodes are at either end of the jumping edge. Jumping should should be avoided in hive plots as it makes the plot aesthetically unpleasing. However, depending upon how you process the data, this condition may arise and hence it is useful to be able to locate jumps.

Value

A summary of the HivePlotData object's key characteristics is printed at the console, followed by the results of any checks set to TRUE. The format of these results is identical to that of plot.list described just below, except for the orphan node check. This is formatted the same as HPD$nodes; see ?HPD for details.

If plot.list = TRUE, a data frame containing a list of the edges to be drawn in a format suitable for troubleshooting a plot. If tex = TRUE as well, the data frame will be in a format suitable for pasting into a LaTeX document. The data frame will contain rows describing each edge to be drawn with the following columns: node 1 id, node 1 axis, node 1 label, node 1 radius, then the same info for node 2, then the edge weight and the edge color.

If orphan.list = TRUE a data frame giving the orphan nodes is returned. If you want both plot.list and orphan.list you have to call this function twice.

Author(s)

Bryan A. Hanson, DePauw University. <hanson@depauw.edu>

Examples

```
set.seed(55)
test <- ranHiveData(nx = 4, ne = 5, desc = "Tiny 4D data set")
out <- sumHPD(test, chk.all = TRUE, plot.list = TRUE)
print(out)
```

Index