Package ‘loon.tourr’

December 18, 2025

Type Package

Title Tour in 'Loon'

Version 0.1.5

Description Implement tour algorithms in interactive graphical system 'loon'.

License GPL-2

Depends R (>= 3.4.0), tcltk, loon (> 1.3.1), tourr, methods,

Imports stats, utils, grDevices, MASS, loon.ggplot, tibble

Suggests class, magrittr, tidyverse, testthat, knitr, rmarkdown, markdown

BugReports `https://github.com/z267xu/loon.tourr/issues`

Encoding UTF-8

RoxygenNote 7.3.1

VignetteBuilder knitr

NeedsCompilation no

Author Zehao Xu [aut, cre],
R. Wayne Oldford [aut]

Maintainer Zehao Xu <`z267xu@gmail.com`>

Repository CRAN

Date/Publication 2025-12-18 06:10:46 UTC

Contents

`l_getPlots.l_tour` *Query a loon widget*

Description

A generic function to query the `loon` (tcl) widget from the given target

Usage

```
## S3 method for class 'l_tour'
l_getPlots(target)
```

Arguments

`target`	a `loon` object

Value

a `loon` widget

Examples

```
if(interactive()) {
  p <- l_tour(iris[, -5])
  l_isLoonWidget(p) # FALSE
  q <- l_getPlots(p)
  l_isLoonWidget(q) # TRUE

  # `l_compound` widget
  p <- l_tour_pairs(tourr::flea[, -7])
  l_isLoonWidget(p) # FALSE
  q <- l_getPlots(p)
  l_isLoonWidget(q) # FALSE
  is(q, "l_compound") # TRUE
}
```

`l_getProjection` *Query the matrix of projection vectors*

Description

Query the matrix of projection vectors

Usage

```
l_getProjection(target, data)
```

Arguments

`target`	A object returned by `l_tour`
`data`	Original data set

Value

a matrix of projection vectors

Examples

```
if(interactive()) {
  dat <- iris[,-5]
  p <- l_tour(dat, color = iris$Species,
              as.l_tour = FALSE)
  # scroll the bar
  proj <- l_getProjection(p, dat)
  projected_object <- as.matrix(dat) %*% proj
  # it will not be exactly the same
  plot(projected_object[,1], projected_object[,2],
       col = hex12tohex6(p['color']))
}
```

`l_layer_callback` *Tour Layer Configuration*

Description

Mainly used in the 2D (or 1D) tour interactive layer configuration

Usage

```
l_layer_callback(target, layer, ...)
```

Arguments

`target`	either a ‘l_tour‘ object or a loon widget
`layer`	the layer need to be modified
`...`	some useful info for the layer configuration (i.e. tours, projections, etc)

Details

It is a `S3` method. The object class is determined by the layer **label**

Value

this callback function does not return any object. As the slider bar is scrolled, for the specified layer, the callback function will be fired and the layer will be configured.

Examples

```
if(interactive() && requireNamespace("tourr")) {
  # 1D tour
  p <- l_tour(iris[, -5], tour = tourr::grand_tour(1L))
  # add layer density
  l <- l_layer(l_getPlots(p),
               stats::density(p['x']),
               label = "density")

  # as we scroll the bar, the density curve does not change
  # unless the following function is executed
  l_layer_callback.density <- function(target, layer, ...) {

      layer <- loon::l_create_handle(c(l_getPlots(target), layer))
      den <- stats::density(target['x'])

      loon::l_configure(layer,
                        x = den$x,
                        y = den$y)

      invisible()
  }
}
```

`l_layer_density2d` *2D density layer*

Description

Two-dimensional kernel density estimation with an axis-aligned bivariate normal kernel

Usage

```
l_layer_density2d(
  widget,
  x,
  y,
  h,
  n = 25L,
  lims = NULL,
  color = "black",
  linewidth = 1,
  nlevels = 10,
  levels = NULL,
  label = "density2d",
  parent = "root",
  index = 0,
```

```
    group = NULL,
    active = TRUE,
    ...
  )
```

Arguments

`widget`	‘loon‘ widget path name as a string
`x`	The coordinates of `x`. See details
`y`	The coordinates of `y`. See details
`h`	vector of bandwidths for x and y directions. Defaults to normal reference bandwidth (see `bandwidth.nrd`). A scalar value will be taken to apply to both directions.
`n`	Number of grid points in each direction. Can be scalar or a length-2 integer vector.
`lims`	The limits of the rectangle covered by the grid as `c(xl, xu, yl, yu)`.
`color`	color of each contour
`linewidth`	the line width
`nlevels`	As described in `grDevices::contourLines`: number of contour levels desired iff `levels` is not supplied.
`levels`	As described in `grDevices::contourLines`: numeric vector of levels at which to draw contour lines.
`label`	label used in the layers inspector
`parent`	parent group layer
`index`	of the newly added layer in its parent group
`group`	separate x vector or y vector into a list by group
`active`	a logical determining whether points appear or not (default is `TRUE` for all points). If a logical vector is given of length equal to the number of points, then it identifies which points appear (`TRUE`) and which do not (`FALSE`).
`...`	other arguments to modify `l_layer_line`.

Value

an `l_layer` widget

Examples

```
if(interactive()) {
p <- l_plot(iris, color = iris$Species)
l <- l_layer_density2d(p)
}
```

`l_layer_hull` *Layer a hull for* `loon`

Description

Creates a layer which is the subset of points lying on the hull (convex or alpha) of the set of points specified.

Usage

```
l_layer_hull(
  widget,
  x,
  y,
  color = "black",
  linewidth = 1,
  label = "hull",
  parent = "root",
  index = 0,
  group = NULL,
  active = TRUE,
  ...
)
```

Arguments

`widget`	‘loon‘ widget path name as a string
`x`	The coordinates of `x`. See details
`y`	The coordinates of `y`. See details
`color`	the line color of each hull
`linewidth`	the line width
`label`	label used in the layers inspector
`parent`	parent group layer
`index`	of the newly added layer in its parent group
`group`	separate x vector or y vector into a list by group
`active`	a logical determining whether points appear or not (default is `TRUE` for all points). If a logical vector is given of length equal to the number of points, then it identifies which points appear (`TRUE`) and which do not (`FALSE`).
`...`	other arguments to modify `l_layer_line`.

Details

Coordinates: the `x` or `y` can be a list or a vector.

- If they are vectors, the argument `group` will be used to set the groups.
- If they are not provided, the `x` will be inherited from the widget

Value

an `l_layer` widget

Examples

```
if(interactive()) {
p <- l_plot(iris, color = iris$Species)
l <- l_layer_hull(p, group = iris$Species)
}
```

`l_layer_trails` *Display tour path with trails*

Description

A 2D tour path with trails

Usage

```
l_layer_trails(
  widget,
  x,
  y,
  xpre,
  ypre,
  color = "black",
  linewidth = 1,
  label = "trails",
  parent = "root",
  index = 0,
  active = TRUE,
  ...
)
```

Arguments

`widget`	‘loon‘ widget path name as a string
`x`	The coordinates of x representing the current state
`y`	The coordinates of y representing the current state
`xpre`	the same length of `x` representing the last state
`ypre`	the same length of `y` representing the last state
`color`	the color of the trail
`linewidth`	the line width
`label`	label used in the layers inspector

`parent`	parent group layer
`index`	of the newly added layer in its parent group
`active`	a logical determining whether points appear or not (default is `TRUE` for all points). If a logical vector is given of length equal to the number of points, then it identifies which points appear (`TRUE`) and which do not (`FALSE`).
`...`	other arguments to modify `l_layer_line`.

Value

an `l_layer` widget

Examples

```
if(interactive()) {
p <- l_tour(iris[, -5], color = iris$Species)
l <- l_layer_trails(p, color = "grey50")
}
```

`l_tour` *Tour in loon*

Description

An interactive tour in loon

Usage

```
l_tour(
  data,
  scaling = c("data", "variable", "observation", "sphere"),
  by = NULL,
  on,
  as.l_tour = TRUE,
  color = loon::l_getOption("color"),
  tour_path = tourr::grand_tour(),
  group = "color",
  start = NULL,
  slicing = FALSE,
  slicingDistance = NULL,
  numOfTours = 30L,
  interpolation = 40L,
  parent = NULL,
  envir = parent.frame(),
  ...
)
```

Arguments

`data`	a data frame with numerical data only
`scaling`	one of 'variable', 'data', 'observation', 'sphere', or 'none' to specify how the data is scaled. See Details
`by`	loon plot can be separated by some variables into multiple panels. This argument can take a `formula`, `n` dimensional state names (see `l_nDimStateNames`) an `n`-dimensional `vector` and `data.frame` or a `list` of same lengths `n` as input.
`on`	if the `x` or `by` is a formula, an optional data frame containing the variables in the `x` or `by`. If the variables are not found in data, they are taken from environment, typically the environment from which the function is called.
`as.l_tour`	return a `l_tour` object; see details
`color`	vector with line colors. Default is given by `l_getOption("color")`.
`tour_path`	tour path generator, defaults to 2d grand tour
`group`	only used for layers. As we scroll the bar, the layers are re-calculated. This argument is used to specify which state is used to set groups (i.e. "color", "linewidth", etc).
`start`	projection to start at, if not specified, uses default associated with tour path
`slicing`	whether to show a sliced scatter plot
`slicingDistance`	the slicing distance that if the distance between points and the projected plane is less than this distance, points will be preserved; else points will be invisible. The default is `NULL` and a suggested value will be given. See details
`numOfTours`	the number of tours
`interpolation`	the steps between two serial projections. The larger the value is, the smoother the transitions would be.
`parent`	a valid Tk parent widget path. When the parent widget is specified (i.e. not `NULL`) then the plot widget needs to be placed using some geometry manager like `tkpack` or `tkplace` in order to be displayed. See the examples below.
`envir`	the `environment` to use.
`...`	named arguments to modify the serialaxes states or layouts, see details.

Details

- tour_path is a tour generator; available tours are `grand_tour`, `dependence_tour`, `frozen_tour`, `guided_tour`, `planned_tour`, and etc
- Argument `as.l_tour`
  - If set to `TRUE`, the function returns an `l_tour` (or an `l_tour_compound`) object. Essentially, this object is a list with the first element being a `loon` (Tcl) widget and the second element a matrix of projection vectors. The advantage of this setup is that the matrix of projection vectors can be easily accessed using the `` `[` `` function (or the `l_cget` function). However, a limitation is that it does not constitute a valid `loon` (Tcl) widget-calling `l_isLoonWidget` would return `FALSE`. Consequently, many of loon's functionalities remain inaccessible.

  - If set to `FALSE`, the function returns either a `loon` (Tcl) widget (where calling `l_isLoonWidget` would return `TRUE`) or an `l_compound` object. In this case, the matrix of projection vectors is not directly accessible from it. However, the `l_getProjection` function can be used to retrieve an estimated matrix of projection vectors.
- The `scaling` state defines how the data is scaled. The axes display 0 at one end and 1 at the other. For the following explanation assume that the data is in a n x p dimensional matrix. The scaling options are then

variable	per column scaling
observation	per row scaling
data	whole matrix scaling
sphere	transforming variables to principal components

- The default `slidingDistance` is suggested by Laa, U., Cook, D., & Valencia, G. (2020). First, find the maximum Euclidean distance of each observation (centralized), say `maxD`. Then, compute the "relative volume" that `vRel` = (`maxD`^(d - 2))/10, where `d` is the dimension of this data set. In the end, the suggested `slidingDistance` is given by `vRel`^(1/(d - 2))

Value

an `l_tour` or an `l_tour_compound` object that one can query the `loon` states and a matrix projection vectors

See Also

`l_getProjection`

Examples

```
if(interactive() && requireNamespace('tourr')) {
  # 2D projection
  fl <- tourr::flea[, 1:6]
  # different scaling will give very different projections
  # in this dataset, scaling 'variable' will give the best separation
  p <- l_tour(fl, scaling = 'variable',
              color = tourr::flea$species)
  l0 <- l_layer_hull(p, group = p["color"],
                     color = "red", linewidth = 4)
  l1 <- l_layer_density2d(p)
  # a `l_tour` object
  class(p)

  # query the matrix of projection vectors
  proj <- p['projection'] # or `l_getProjection(p)`
  # suppose the scaling is still 'observation'
  new_xy <- as.matrix(
    loon::l_getScaledData(data = fl,
                          scaling = 'observation')) %*%
    proj
  plot(new_xy, xlab = "V1", ylab = "V2",
       col = loon::hex12tohex6(p['color']))
```

```
  # A higher dimension projection
  # turn the `tour` to 4 dimensional space
  s <- l_tour(fl, color = tourr::flea$species,
              scaling = "observation",
              tour_path = tourr::grand_tour(4L))

  # set `as.l_tour` FALSE
  p <- l_tour(fl, scaling = 'observation',
              color = tourr::flea$species)
  class(p)
  ## ERROR
  ## p["projection"]

  # query the estimated matrix of projection vectors
  l_getProjection(p)

  ##### facet by region
  olive <- tourr::olive
  p <- with(olive, l_tour(olive[, -c(1, 2)],
                          by = region,
                          color = area))
}
```

`tour_pairs` *Tour Pairs Plot*

Description

A nD tour path with a scatterplot matrix (the default tour is a 4D tour; by setting ‘tour_path‘ to modify the dimension)

Usage

```
l_tour_pairs(
  data,
  scaling = c("data", "variable", "observation", "sphere"),
  tour_path = tourr::grand_tour(4L),
  numOfTours = 30L,
  interpolation = 40L,
  as.l_tour = TRUE,
  connectedScales = c("none", "cross"),
  linkingGroup,
  linkingKey,
  showItemLabels = TRUE,
  itemLabel,
  showHistograms = FALSE,
  histLocation = c("edge", "diag"),
  histHeightProp = 1,
```

```
  histArgs = list(),
  showSerialAxes = FALSE,
  serialAxesArgs = list(),
  color = "grey60",
  group = "color",
  start = NULL,
  parent = NULL,
  span = 10L,
  envir = parent.frame(),
  ...
)
```

Arguments

`data`	a data frame with numerical data only
`scaling`	one of ’variable’, ’data’, ’observation’, ’sphere’, or ’none’ to specify how the data is scaled. See Details
`tour_path`	tour path generator, defaults to 2d grand tour
`numOfTours`	the number of tours
`interpolation`	the steps between two serial projections. The larger the value is, the smoother the transitions would be.
`as.l_tour`	return a `l_tour` object; see details
`connectedScales`	Determines how the scales of the panels are to be connected. • `"cross"`: only the scales in the same row and the same column are connected; • `"none"`: neither "x" nor "y" scales are connected in any panels.
`linkingGroup`	string giving the linkingGroup for all plots. If missing, a default `linkingGroup` will be determined from deparsing the `data`.
`linkingKey`	a vector of strings to provide a linking identity for each row of the `data` data.frame. If missing, a default `linkingKey` will be `0:(nrows(data)-1)`.
`showItemLabels`	TRUE, logical indicating whether its itemLabel pops up over a point when the mouse hovers over it.
`itemLabel`	a vector of strings to be used as pop up information when the mouse hovers over a point. If missing, the default `itemLabel` will be the `row.names(data)`.
`showHistograms`	logical (default FALSE) to show histograms of each variable or not
`histLocation`	one "edge" or "diag", when showHistograms = TRUE
`histHeightProp`	a positive number giving the height of the histograms as a proportion of the height of the scatterplots
`histArgs`	additional arguments to modify the ‘l_hist‘ states
`showSerialAxes`	logical (default FALSE) indication of whether to show a serial axes plot in the bottom left of the pairs plot (or not)
`serialAxesArgs`	additional arguments to modify the ‘l_serialaxes‘ states

`color`	vector with line colors. Default is given by `l_getOption("color")`.
`group`	only used for layers. As we scroll the bar, the layers are re-calculated. This argument is used to specify which state is used to set groups (i.e. "color", "linewidth", etc).
`start`	projection to start at, if not specified, uses default associated with tour path
`parent`	a valid Tk parent widget path. When the parent widget is specified (i.e. not `NULL`) then the plot widget needs to be placed using some geometry manager like `tkpack` or `tkplace` in order to be displayed. See the examples below.
`span`	How many column/row occupies for each widget
`envir`	the `environment` to use.
`...`	named arguments to modify the serialaxes states or layouts, see details.

Value

an `l_tour_compound` object that one can query the `loon` states and a matrix projection vectors

See Also

`l_pairs`, `l_tour`

Examples

```
if(interactive() && requireNamespace('tourr')) {
  # q is a `l_pairs` object
  q <- l_tour_pairs(olive[, -c(1:2)],
                    color = olive$region)
  # query the matrix of projection vectors
  proj <- q["projection"]

  # query the `l_compound` widget
  lc <- l_getPlots(q)
  # pack the `density2d` layers
  layer_pack <- lapply(lc, function(w) l_layer_density2d(w))

  #### set `as.l_tour = FALSE`
  # q is a `l_pairs` object
  q <- l_tour_pairs(tourr::flea[, 1:6],
                    as.l_tour = FALSE,
                    color = tourr::flea$species,
                    showHistogram = TRUE,
                    showSerialAxes = TRUE)

  # proj <- q["projection"] # Return a list of `NA`
  # query estimated matrix of projection vectors
  proj <- l_getProjection(q, tourr::flea[, 1:6])
}
```

Index