*# AIM*

*# ggplot2 scatterplot with the possibility to overlay dots from up to 3 different data frames (-> three different legends) and lines from up to 3 different data frames (-> three different legends) -> up to 6 overlays totally*

*# for ggplot2 specifications, see: https://ggplot2.tidyverse.org/articles/ggplot2-specs.html*

*# WARNINGS*

*# rows containing NA in data1[, c(y, categ)] will be removed before processing, with a warning (see below)*

*# ARGUMENTS*

*# data1: a dataframe compatible with ggplot2, or a list of data frames*

*# x: character string of the data1 column name for x-axis. If data1 is a list, then x must be a list of character strings, of same size as data1, with compartment 1 related to compartment 1 of data1, etc. Write NULL for each "geom\_hline" in geom argument*

*# y: character string of the data1 column name for y-axis. If data1 is a list, then y must be a list of character strings, of same size as data1, with compartment 1 related to compartment 1 of data1, etc. Write NULL for each "geom\_vline" in geom argument*

*# categ: character string of the data1 column name for categories. If categ == NULL, no categories -> no legend displayed. If data1 is a list, then categ must be a list of character strings, of same size as data1, with compartment 1 related to compartment 1 of data1, etc. Some of the list compartments can be NULL, and other not*

*# legend.name: character string of the legend title. If legend.name == NULL and categ != NULL, then legend.name <- categ. If data1 is a list, then legend.name must be a list of character strings, of same size as data1, with compartment 1 related to compartment 1 of data1, etc. Some of the list compartments can be NULL, and other not*

*# color: vector of character string of the colors of categ arguments. If color == NULL, default colors of ggplot2. If non null, it can be either: (1) a single color string (all the dots of the corresponding data1 will have this color, whatever categ NULL or not), (2) if categ non null, a vector of string colors, one for each class of categ (each color will be associated according to the alphabetical order of categ classes), (3) if categ non null, a vector or factor of string colors, like if it was one of the column of data1 data frame (beware: a single color per class of categ and a single class of categ per color must be respected). Integers are also accepted instead of character strings, as long as above rules about length are respected. Integers will be processed by fun\_gg\_palette() using the max integer value among all the integers in color. If data1 is a list, then color must be a list of character strings or integers, of same size as data1, with compartment 1 related to compartment 1 of data1, etc. Some of the compartments can be NULL. In that case, a different grey color will be used for each NULL compartment*

*# geom: character string of the kind of plot. Either "geom\_point" (scatterplot), "geom\_line" (coordinates plotted then line connection from the lowest to highest coordinates), "geom\_path" (line connection respecting the order in data1), "geom\_hline" (horizontal line) or "geom\_vline" (vertical line). BEWARE: for "geom\_hline" or "geom\_vline", (1) x or y argument must be NULL, respectively, (2) x.lim or y.lim argument must NOT be NULL, respectively, if only these kind of lines are drawn (if other geom present, then x.lim = NULL and y.lim = NULL will generate x.lim and y.lim defined by these other geom, which is not possible with "geom\_hline" or "geom\_vline"), (3) the function will draw n lines for n values in the x argument column name of the data1 data frame. If several colors required, the categ argument must be specified and the corresponding categ column name must exist in the data1 data frame with a different class name for each row. If data1 is a list, then geom must be a list of character strings, of same size as data1, with compartment 1 related to compartment 1 of data1, etc.*

*# alpha: numeric value (from 0 to 1) of transparency. If data1 is a list, then alpha must be a list of numeric value, of same size as data1, with compartment 1 related to compartment 1 of data1, etc.*

*# dot.size: numeric value of point size*

*# line.size: numeric value of line size*

*# x.lim: 2 numeric values for x-axis range. If NULL, range of x in data1. Order of the 2 values matters (for inverted axis). BEWARE: values of the x.lim must be already in the corresponding log if x.log argument is not "no" (see below)*

*# x.lab: a character string or expression for x-axis legend. If NULL, x of the first data frame in data1. Warning message if the elements in x are different between data frames in data1*

*# x.log: Either "no" (values in the x argument column of the data1 data frame are not log), "log2" (values in the x argument column of the data1 data frame are log2 transformed) or "log10" (values in the x argument column of the data1 data frame are log10 transformed). BEWARE: the function does not tranform the data, but just displays ticks in a log scale manner. Thus, negative or zero values are allowed*

*# x.tick.nb: approximate number of desired label values on the x-axis (n argument of the fun\_scale() function). If NULL, the number is managed by ggplot2*

*# x.inter.tick.nb: number of desired secondary ticks between main ticks. Not considered if x.log is other than "no". In that case, play with the x.lim and x.tick.nb arguments. IF NULL, no secondary ticks*

*# x.include.zero: logical. Does x.lim range include 0? Ok even if x.log == TRUE because x, and thus x.lim, must already be log transformed values*

*# x.left.extra.margin: single proportion (between 0 and 1) indicating if extra margins must be added to x.lim. If different from 0, add the range of the axis \* x.left.extra.margin (e.g., abs(x.lim[2] - x.lim[1]) \* x.left.extra.margin) to the left of x-axis*

*# x.right.extra.margin: idem as x.left.extra.margin but to the bottom of x-axis*

*# x.text.angle: integer value of the text angle for the x-axis labels. Positive values for counterclockwise rotation: 0 for horizontal, 90 for vertical, 180 for upside down etc. Negative values for clockwise rotation: 0 for horizontal, -90 for vertical, -180 for upside down etc.*

*# y.lim: 2 numeric values for y-axis range. If NULL, range of y in data1. Order of the 2 values matters (for inverted axis). BEWARE: values of the y.lim must be already in the corresponding log if y.log argument is not "no" (see below)*

*# y.lab: a character string or expression for y-axis legend. If NULL, y of the first data frame in data1. Warning message if the elements in y are different between data frames in data1*

*# y.log: Either "no" (values in the y argument column of the data1 data frame are not log), "log2" (values in the y argument column of the data1 data frame are log2 transformed) or "log10" (values in the y argument column of the data1 data frame are log10 transformed). BEWARE: the function does not tranform the data, but just displays ticks in a log scale manner. Thus, negative or zero values are allowed*

*# y.tick.nb: approximate number of desired label values on the y-axis (n argument of the fun\_scale() function). If NULL, the number is managed by ggplot2*

*# y.inter.tick.nb: number of desired secondary ticks between main ticks. Not considered if y.log is other than "no". In that case, play with the y.lim and y.tick.nb arguments. IF NULL, no secondary ticks*

*# y.include.zero: logical. Does y.lim range include 0? Ok even if y.log == TRUE because y, and thus y.lim, must already be log transformed values*

*# y.left.extra.margin: single proportion (between 0 and 1) indicating if extra margins must be added to y.lim. If different from 0, add the range of the axis \* y.left.extra.margin (e.g., abs(y.lim[2] - y.lim[1]) \* y.left.extra.margin) to the left of y-axis*

*# y.right.extra.margin: idem as y.left.extra.margin but to the bottom of y-axis*

*# y.text.angle: integer value of the text angle for the y-axis labels. Positive values for counterclockwise rotation: 0 for horizontal, 90 for vertical, 180 for upside down etc. Negative values for clockwise rotation: 0 for horizontal, -90 for vertical, -180 for upside down etc.*

*# text.size: numeric value of the size of the (1) axis numbers and axis legends and (2) texts in the graphic legend*

*# title: character string of the graph title*

*# title.text.size: numeric value of the title size (in points)*

*# show.legend: logical. Show legend? Not considered if categ argument is NULL, because this already generate no legend*

*# legend.sector.width: single proportion (between 0 and 1) indicating the relative width of the legend sector (on the right of the plot) relative to the width of the plot. Value 1 means that the window device width is split in 2, half for the plot and half for the legend. Write NULL to inactivate the legend sector. In such case, ggplot2 will manage the room required for the legend display, meaning that the width of the plotting region can vary between graphs, depending on the text in the legend*

*# classic: logical. Use the classic theme (article like) more than the standard ggplot2 theme?*

*# grid: logical. Draw horizontal and vertical lines in the background to better read the values? Not considered if classic == FALSE*

*# raster: logical. Dots in raster mode? If FALSE, dots from each geom\_point from geom argument are in vectorial mode (bigger pdf and long to display if millions of dots). If TRUE, dots from each geom\_point from geom argument are in matricial mode (smaller pdf and easy display if millions of dots, but long to generate the layer). If TRUE, the plot region will be square to avoid a bug in fun\_gg\_point\_rast(). If TRUE, solve the transparency problem with some GUI. Overriden by vectorial.limit if non NULL*

*# device.ratio: prop. Ignored if raster == FALSE. ADD ALSO IN NO LEGEND SCATTER*

*# vectorial.limit: positive integer value indicating the limit of the dot number above which geom\_point from geom argument switch from vectorial mode to raster mode (see the raster argument). If any layer is raster, then the region plot will be square to avoid a bug in fun\_gg\_point\_rast(). Inactive the raster argument if non NULL*

*# return: logical. Return the graph info?*

*# plot: logical. Plot the graphic? If FALSE and return argument is TRUE, graphical parameters and associated warnings are provided without plotting*

*# add: character string allowing to add more ggplot2 features (dots, lines, themes, etc.). BEWARE: (1) must start with "+" just after the simple or double opening quote (no space, end of line, carriage return, etc., allowed), (2) must finish with ")" just before the simple or double closing quote (no space, end of line, carriage return, etc., allowed) and (3) each function must be preceded by "ggplot2::" (for instance: "ggplot2::coord\_flip()). If the character string contains the "ggplot2::theme" string, then internal ggplot2 theme() and theme\_classic() functions will be inactivated to be reused by add. BEWARE: handle this argument with caution since added functions can create conflicts with the preexisting internal ggplot2 functions. Not considered if NULL*

*# warn.print: logical. Print warnings at the end of the execution? If TRUE, no print if no warning message generated*

*# lib.path: character string indicating the absolute path of the required packages (see below). if NULL, the function will use the R library default folders*

*# REQUIRED PACKAGES*

*# ggplot2*

*# if raster plots are drawn (see the raster and vectorial.limit arguments):*

*# Cairo*

*# grid*

*# REQUIRED FUNCTIONS FROM CUTE\_LITTLE\_R\_FUNCTION*

*# fun\_gg\_palette()*

*# fun\_gg\_point\_rast()*

*# fun\_pack()*

*# fun\_check()*

*# RETURN*

*# a scatter plot is plot argument is TRUE*

*# a list of the graph info if return argument is TRUE:*

*# $data: the graphic info coordinates*

*# $removed.row.nb: a list of the removed rows numbers in data frames (because of NA). NULL if no row removed*

*# $removed.rows: a list of the removed rows in data frames (because of NA). NULL if no row removed*

*# $axes: the x-axis and y-axis info*

*# $warn: the warning messages. Use cat() for proper display. NULL if no warning*

*# EXAMPLES*

*## NICE REPRESENTATION*

*# set.seed(1) ; obs1 <- data.frame(km = rnorm(1000, 10, 3), time = rnorm(1000, 10, 3), group1 = rep(c("A1", "A2"), 500)) ; obs2 <-data.frame(km = rnorm(1000, 15, 3), time = rnorm(1000, 15, 3), group2 = rep(c("G1", "G2"), 500)) ; set.seed(NULL) ; obs1$L1$km[2:3] <- NA ; fun\_gg\_scatter(data1 = list(L1 = obs1, L2 = obs2), x = list(L1 = "km", L2 = "km"), y = list(L1 = "time", L2 = "time"), categ = list(L1 = "group1", L2 = "group2"), legend.name = NULL, color = list(L1 = 4:5, L2 = 7:8), geom = list(L1 = "geom\_point", L2 = "geom\_point"), alpha = list(L1 = 0.5, L2 = 0.5), dot.size = 3, line.size = 0.5, x.lim = c(1, 25), x.lab = "KM", x.log = "no", x.tick.nb = 10, x.inter.tick.nb = 1, x.left.extra.margin = 0, x.right.extra.margin = 0, y.lim = c(1, 25), y.lab = expression(paste("TIME (", 10^-20, " s)")), y.log = "log10", y.tick.nb = 5, y.top.extra.margin = 0, y.bottom.extra.margin = 0, y.include.zero = TRUE, classic = TRUE)*

*## SINGLE GEOMETRIC LAYER*

*### simple example (1) of scatter plot using the classical writting*

*# obs1 <- data.frame(km = 2:7, time = (2:7)^2, group = c("A", "A", "A", "B", "B", "B")) ; obs1 ; fun\_gg\_scatter(data1 = obs1, x = "km", y = "time")*

*### simple example (2) of scatter plot, identical to (1) but using the list writting. Here, a list of one compartment, systematically named L1, is provided to the data1, x, y, categ, geom and alpha. Contrary to example (1), the geom and alpha argument have to be included because the default value are not lists (if data1 is a list, all the x, y, categ, legend.name, color, geom and alpha must also be list if non NULL)*

*# obs1 <- data.frame(km = 2:7, time = (2:7)^2, group = c("A", "A", "A", "B", "B", "B")) ; obs1 ; fun\_gg\_scatter(data1 = list(L1 = obs1), x = list(L1 = "km"), y = list(L1 = "time"), geom = list(L1 = "geom\_point"), alpha = list(L1 = 0.5))*

*### color of dots. Example (1) using the classical writting*

*# obs1 <- data.frame(km = 2:7, time = (2:7)^2, group = c("A", "A", "A", "B", "B", "B")) ; obs1 ; fun\_gg\_scatter(data1 = obs1, x = "km", y = "time", color = "blue")*

*### color of dots. Example (2) using the list writting*

*# obs1 <- data.frame(km = 2:7, time = (2:7)^2, group = c("A", "A", "A", "B", "B", "B")) ; obs1 ; fun\_gg\_scatter(data1 = list(L1 = obs1), x = list(L1 = "km"), y = list(L1 = "time"), color = list(L1 = "blue"), geom = list(L1 = "geom\_point"), alpha = list(L1 = 1))*

*### From here, classical writting is use for single element in data1 and list writting otherwise*

*### color of dots. Example (3) when dots are in different categories. Note that categ argument controls the legend display*

*# obs1 <- data.frame(km = 2:7, time = (2:7)^2, group = c("A", "A", "A", "B", "B", "B")) ; obs1 ; fun\_gg\_scatter(data1 = obs1, x = "km", y = "time", categ = "group")*

*### color of dots. Example (4) when dots are in different categories. A single color mentionned is applied to all the dots*

*# obs1 <- data.frame(km = 2:7, time = (2:7)^2, group = c("A", "A", "A", "B", "B", "B")) ; obs1 ; fun\_gg\_scatter(data1 = obs1, x = "km", y = "time", categ = "group", color = "coral")*

*### color of dots. Example (5) when dots are in different categories. Numbers can be used if ggplot2 colors are desired*

*# obs1 <- data.frame(km = 2:7, time = (2:7)^2, group = c("A", "A", "A", "B", "B", "B")) ; obs1 ; fun\_gg\_scatter(data1 = obs1, x = "km", y = "time", categ = "group", color = 2)*

*### color of dots. Example (6) when dots are in different categories, with one color per category (try also color = 2:1)*

*# obs1 <- data.frame(km = 2:7, time = (2:7)^2, group = c("A", "A", "A", "B", "B", "B")) ; obs1 ; fun\_gg\_scatter(data1 = obs1, x = "km", y = "time", categ = "group", color = c("coral", "green"))*

*### color of dots. Example (7) when dots are in different categories, with colors as a data frame column. BEWARE: one color per category must be respected (try also numbers)*

*# obs1 <- data.frame(km = 2:7, time = (2:7)^2, group = c("A", "A", "A", "B", "B", "B"), col = rep(c("coral", "green"), each = 3)) ; obs1 ; fun\_gg\_scatter(data1 = obs1, x = "km", y = "time", categ = "group", color = obs1$col)*

*### color of dots. Example (8) when dots are in different categories, with colors as a data frame column. Easiest way (ggplot2 colors)*

*# obs1 <- data.frame(km = 2:7, time = (2:7)^2, group = c("A", "A", "A", "B", "B", "B")) ; obs1 ; fun\_gg\_scatter(data1 = obs1, x = "km", y = "time", categ = "group", color = as.numeric(obs1$group))*

*### legend name*

*# obs1 <- data.frame(km = 2:7, time = (2:7)^2, group = c("A", "A", "A", "B", "B", "B")) ; obs1 ; fun\_gg\_scatter(data1 = obs1, x = "km", y = "time", categ = "group", legend.name = "CLASSES")*

*### different geom features. Example (1) with geom\_line kind of lines*

*# obs1 <- data.frame(km = c(1, 3, 2, 6, 4, 5), time = c(1, 3, 2, 6, 4, 5)^2, group = c("A", "A", "A", "B", "B", "B")) ; obs1 ; fun\_gg\_scatter(data1 = obs1, x = "km", y = "time", geom = "geom\_line", categ = "group")*

*### different geom features. Example (2) with geom\_path kind of lines (see the difference with (1))*

*# obs1 <- data.frame(km = c(1, 3, 2, 6, 4, 5), time = c(1, 3, 2, 6, 4, 5)^2, group = c("A", "A", "A", "B", "B", "B")) ; obs1 ; fun\_gg\_scatter(data1 = obs1, x = "km", y = "time", geom = "geom\_path", categ = "group")*

*### different geom features. Example (3) with geom\_hline kind of lines. Fake\_y y-axis name by default because y argument must be NULL (see y.lab argument below to change this)*

*# obs1 <- data.frame(km = 1:2, time = (1:2)^2, group = c("A", "B")) ; obs1 ; fun\_gg\_scatter(data1 = obs1, x = NULL, y = "km", geom = "geom\_hline", categ = "group", x.lim = c(1,10))*

*### different geom features. Example (4) with geom\_vline kind of lines. Fake\_y y-axis name by default because y argument must be NULL (see y.lab argument below to change this)*

*# obs1 <- data.frame(km = 1:2, time = (1:2)^2, group = c("A", "B")) ; obs1 ; fun\_gg\_scatter(data1 = obs1, x = "km", y = NULL, geom = "geom\_vline", categ = "group", y.lim = c(1,10))*

*## MULTI GEOMETRIC LAYERS*

*### Note that in subsequent examples, names of list compartments are systematically referred to as L1, L2, etc., to show the correspondence between the arguments data1, x, y, categ, etc.*

*### single layer (as examples above)*

*# set.seed(1) ; obs1 <- data.frame(km = rnorm(1000, 22, 3), time = rnorm(1000, 22, 3)) ; set.seed(NULL) ; fun\_gg\_scatter(data1 = list(L1 = obs1), x = list(L1 = "km"), y = list(L1 = "time"), geom = list(L1 = "geom\_point"), alpha = list(L1 = 0.5))*

*### simple example of two layers*

*# set.seed(1) ; obs1 <- data.frame(km = rnorm(1000, 22, 3), time = rnorm(1000, 22, 3)) ; obs2 <-data.frame(km = rnorm(1000, 30, 3), time = rnorm(1000, 30, 3)) ; set.seed(NULL) ; fun\_gg\_scatter(data1 = list(L1 = obs1, L2 = obs2), x = list(L1 = "km", L2 = "km"), y = list(L1 = "time", L2 = "time"), geom = list(L1 = "geom\_point", L2 = "geom\_point"), alpha = list(L1 = 0.5, L2 = 0.5))*

*### color of dots. Example (1)*

*# set.seed(1) ; obs1 <- data.frame(km = rnorm(1000, 22, 3), time = rnorm(1000, 22, 3)) ; obs2 <-data.frame(km = rnorm(1000, 30, 3), time = rnorm(1000, 30, 3)) ; set.seed(NULL) ; fun\_gg\_scatter(data1 = list(L1 = obs1, L2 = obs2), x = list(L1 = "km", L2 = "km"), y = list(L1 = "time", L2 = "time"), geom = list(L1 = "geom\_point", L2 = "geom\_point"), alpha = list(L1 = 0.5, L2 = 0.5), color = list(L1 = "coral", L2 = "green"))*

*### color of dots. Example (2) of the legend display. The categ argument must be supplied. Make a fake categorical colum in the data frame if necessary (as in this example). The categ argument triggers the legend display. The legend.name argument is used to remove the legend title of each layer*

*# set.seed(1) ; obs1 <- data.frame(km = rnorm(1000, 22, 3), time = rnorm(1000, 22, 3), group1 = "GROUP1") ; obs2 <-data.frame(km = rnorm(1000, 30, 3), time = rnorm(1000, 30, 3), group2 = "GROUP2") ; set.seed(NULL) ; fun\_gg\_scatter(data1 = list(L1 = obs1, L2 = obs2), x = list(L1 = "km", L2 = "km"), y = list(L1 = "time", L2 = "time"), categ = list(L1 = "group1", L2 = "group2"), legend.name = list(L1 = NULL, L2 = NULL), geom = list(L1 = "geom\_point", L2 = "geom\_point"), alpha = list(L1 = 0.5, L2 = 0.5), color = list(L1 = "coral", L2 = "green"))*

*### color of dots. Example (3) when dots are in different categories (default colors)*

*# set.seed(1) ; obs1 <- data.frame(km = rnorm(1000, 22, 3), time = rnorm(1000, 22, 3), group1 = rep(c("A1", "A2"), each = 500)) ; obs2 <-data.frame(km = rnorm(1000, 30, 3), time = rnorm(1000, 30, 3), group2 = rep(c("G1", "G2"), each = 500)) ; set.seed(NULL) ; fun\_gg\_scatter(data1 = list(L1 = obs1, L2 = obs2), x = list(L1 = "km", L2 = "km"), y = list(L1 = "time", L2 = "time"), , categ = list(L1 = "group1", L2 = "group2"), geom = list(L1 = "geom\_point", L2 = "geom\_point"), alpha = list(L1 = 0.5, L2 = 0.5))*

*### color of dots. Example (3) when dots are in different categories. A single color mentionned per layer is applied to all the dots of the layer*

*# set.seed(1) ; obs1 <- data.frame(km = rnorm(1000, 22, 3), time = rnorm(1000, 22, 3), group1 = rep(c("A1", "A2"), each = 500)) ; obs2 <-data.frame(km = rnorm(1000, 30, 3), time = rnorm(1000, 30, 3), group2 = rep(c("G1", "G2"), each = 500)) ; set.seed(NULL) ; fun\_gg\_scatter(data1 = list(L1 = obs1, L2 = obs2), x = list(L1 = "km", L2 = "km"), y = list(L1 = "time", L2 = "time"), , categ = list(L1 = "group1", L2 = "group2"), geom = list(L1 = "geom\_point", L2 = "geom\_point"), alpha = list(L1 = 0.5, L2 = 0.5), color = list(L1 = "coral", L2 = "green"))*

*### color of dots. Example (5) when dots are in different categories, with one color per category in each layer*

*# set.seed(1) ; obs1 <- data.frame(km = rnorm(1000, 22, 3), time = rnorm(1000, 22, 3), group1 = rep(c("A1", "A2"), each = 500)) ; obs2 <-data.frame(km = rnorm(1000, 30, 3), time = rnorm(1000, 30, 3), group2 = rep(c("G1", "G2"), each = 500)) ; set.seed(NULL) ; fun\_gg\_scatter(data1 = list(L1 = obs1, L2 = obs2), x = list(L1 = "km", L2 = "km"), y = list(L1 = "time", L2 = "time"), , categ = list(L1 = "group1", L2 = "group2"), geom = list(L1 = "geom\_point", L2 = "geom\_point"), alpha = list(L1 = 0.5, L2 = 0.5), color = list(L1 = c("coral", "blue"), L2 = c("green", "black")))*

*### color of dots. Example (4) when dots are in different categories. Numbers can be used if ggplot2 colors are desired*

*# set.seed(1) ; obs1 <- data.frame(km = rnorm(1000, 22, 3), time = rnorm(1000, 22, 3), group1 = rep(c("A1", "A2"), each = 500)) ; obs2 <-data.frame(km = rnorm(1000, 30, 3), time = rnorm(1000, 30, 3), group2 = rep(c("G1", "G2"), each = 500)) ; set.seed(NULL) ; fun\_gg\_scatter(data1 = list(L1 = obs1, L2 = obs2), x = list(L1 = "km", L2 = "km"), y = list(L1 = "time", L2 = "time"), , categ = list(L1 = "group1", L2 = "group2"), geom = list(L1 = "geom\_point", L2 = "geom\_point"), alpha = list(L1 = 0.5, L2 = 0.5), color = list(L1 = 1:2, L2 = c(4, 7)))*

*### color of dots. Example (7) when dots are in different categories, with colors as a data frame column. BEWARE: one color per category must be respected (try also numbers). BEWARE: in color argument, if the column of the data frame does not exist, color can be still displayed (L2 = obs2$notgood is equivalent to L2 = NULL). Such situation is reported in the warning messages (see below)*

*# set.seed(1) ; obs1 <- data.frame(km = rnorm(1000, 22, 3), time = rnorm(1000, 22, 3), group1 = rep(c("A1", "A2"), each = 500), col1 = rep(c("coral", "blue"), each = 500)) ; obs2 <-data.frame(km = rnorm(1000, 30, 3), time = rnorm(1000, 30, 3), group2 = rep(c("G1", "G2"), each = 500), col2 = rep(c("green", "black"), each = 500)) ; set.seed(NULL) ; fun\_gg\_scatter(data1 = list(L1 = obs1, L2 = obs2), x = list(L1 = "km", L2 = "km"), y = list(L1 = "time", L2 = "time"), , categ = list(L1 = "group1", L2 = "group2"), geom = list(L1 = "geom\_point", L2 = "geom\_point"), alpha = list(L1 = 0.5, L2 = 0.5), color = list(L1 = obs1$col1, L2 = obs2$col2))*

*### color of dots. Example (8) when dots are in different categories, with colors as a data frame column. Easiest way is not recommended with mutiple layers*

*# set.seed(1) ; obs1 <- data.frame(km = rnorm(1000, 22, 3), time = rnorm(1000, 22, 3), group1 = rep(c("A1", "A2"), each = 500), col1 = rep(c("coral", "blue"), each = 500)) ; obs2 <-data.frame(km = rnorm(1000, 30, 3), time = rnorm(1000, 30, 3), group2 = rep(c("G1", "G2"), each = 500), col2 = rep(c("green", "black"), each = 500)) ; set.seed(NULL) ; fun\_gg\_scatter(data1 = list(L1 = obs1, L2 = obs2), x = list(L1 = "km", L2 = "km"), y = list(L1 = "time", L2 = "time"), , categ = list(L1 = "group1", L2 = "group2"), geom = list(L1 = "geom\_point", L2 = "geom\_point"), alpha = list(L1 = 0.5, L2 = 0.5), color = list(L1 = as.numeric(obs1$group1), L2 = as.numeric(obs2$group2)))*

*### legend name*

*# set.seed(1) ; obs1 <- data.frame(km = rnorm(1000, 22, 3), time = rnorm(1000, 22, 3), group1 = rep(c("A1", "A2"), each = 500)) ; obs2 <-data.frame(km = rnorm(1000, 30, 3), time = rnorm(1000, 30, 3), group2 = rep(c("G1", "G2"), each = 500)) ; set.seed(NULL) ; fun\_gg\_scatter(data1 = list(L1 = obs1, L2 = obs2), x = list(L1 = "km", L2 = "km"), y = list(L1 = "time", L2 = "time"), , categ = list(L1 = "group1", L2 = "group2"), legend.name = list(L1 = "CLASS A", L2 = "CLASS G"), geom = list(L1 = "geom\_point", L2 = "geom\_point"), alpha = list(L1 = 0.5, L2 = 0.5))*

*### different geom features. Example (1) with 5 layers. Note that order in data1 defines the overlay order (from below to above) and the order in the legend (from top to bottom)*

*# set.seed(1) ; obs1 <- data.frame(km = rnorm(1000, 22, 3), time = rnorm(1000, 22, 3), group1 = rep(c("A1", "A2"), each = 500)) ; obs2 <-data.frame(km = rnorm(1000, 30, 3), time = rnorm(1000, 30, 3), group2 = rep(c("G1", "G2"), each = 500)) ; set.seed(NULL) ; obs3 <- data.frame(time = c(29, 31), group3 = c("HORIZ.THRESHOLD.1", "HORIZ.THRESHOLD.2")) ; obs4 <- data.frame(km = 26, group4 = "VERTIC.THRESHOLD") ; obs5 <- data.frame(km = seq(1, 100, 0.1), time = 7\*seq(1, 100, 0.1)^0.5, group5 = "FUNCTION") ; fun\_gg\_scatter(data1 = list(L1 = obs1, L2 = obs2, L3 = obs3, L4 = obs4, L5 = obs5), x = list(L1 = "km", L2 = "km", L3 = NULL, L4 = "km", L5 = "km"), y = list(L1 = "time", L2 = "time", L3 = "time", L4 = NULL, L5 = "time"), categ = list(L1 = "group1", L2 = "group2", L3 = "group3", L4 = "group4", L5 = "group5"), geom = list(L1 = "geom\_point", L2 = "geom\_point", L3 = "geom\_hline", L4 = "geom\_vline", L5 = "geom\_line"), alpha = list(L1 = 0.5, L2 = 0.5, L3 = 0.5, L4 = 0.5, L5 = 0.5), x.lim = c(10, 40), y.lim = c(10, 40), classic = TRUE, line.size = 0.75)*

*### layer transparency. One transparency defined by layer (from 0 invisible to 1 opaque). Note that for lines, transparency in not applied in the legend to prevent a ggplot2 bug on windows (https://github.com/tidyverse/ggplot2/issues/2452)*

*# set.seed(1) ; obs1 <- data.frame(km = rnorm(1000, 22, 3), time = rnorm(1000, 22, 3), group1 = rep(c("A1", "A2"), each = 500)) ; obs2 <-data.frame(km = rnorm(1000, 30, 3), time = rnorm(1000, 30, 3), group2 = rep(c("G1", "G2"), each = 500)) ; set.seed(NULL) ; fun\_gg\_scatter(data1 = list(L1 = obs1, L2 = obs2), x = list(L1 = "km", L2 = "km"), y = list(L1 = "time", L2 = "time"), , categ = list(L1 = "group1", L2 = "group2"), geom = list(L1 = "geom\_point", L2 = "geom\_point"), alpha = list(L1 = 1, L2 = 0.1))*

*### other different example of mutiple geom features are shown in the fun\_segmentation function*

*## OTHER GRAPHIC ARGUMENTS*

*### dot size (line.size argument controls size of lines)*

*# obs1 <- data.frame(km = 2:7, time = (2:7)^2, group = c("A", "A", "A", "B", "B", "B")) ; obs1 ; fun\_gg\_scatter(data1 = obs1, x = "km", y = "time", dot.size = 5)*

*### axis management: examples are shown for x-axis but are identical for y-axis*

*### x-axis limits. Example (1)*

*# obs1 <- data.frame(km = 2:7, time = (2:7)^2, group = c("A", "A", "A", "B", "B", "B")) ; obs1 ; fun\_gg\_scatter(data1 = obs1, x = "km", y = "time", x.lim = c(-1, 25))*

*### x-axis limits. Example (2) showing that order matters in y.lim argument*

*# obs1 <- data.frame(km = 2:7, time = (2:7)^2, group = c("A", "A", "A", "B", "B", "B")) ; obs1 ; fun\_gg\_scatter(data1 = obs1, x = "km", y = "time", x.lim = c(25, -1))*

*### log scale. Example (1). BEWARE: x column must be log, otherwise incoherent scale (see below warning message with the return argument)*

*# obs1 <- data.frame(km = 2:7, time = (2:7)^2, group = c("A", "A", "A", "B", "B", "B")) ; obs1 ; fun\_gg\_scatter(data1 = obs1, x = "km", y = "time", x.log = "log10")*

*### log scale. Example (2). BEWARE: values of the x.lim must be in the corresponding log*

*# obs1 <- data.frame(km = 2:7, time = (2:7)^2, group = c("A", "A", "A", "B", "B", "B")) ; obs1 ; fun\_gg\_scatter(data1 = obs1, x = "km", y = "time", x.log = "log10", x.lim = c(1, 10))*

*### tick number. Example (1). Note that the final number shown is approximate*

*# obs1 <- data.frame(km = 2:7, time = (2:7)^2, group = c("A", "A", "A", "B", "B", "B")) ; obs1 ; fun\_gg\_scatter(data1 = obs1, x = "km", y = "time", x.tick.nb = 6)*

*### tick number. Example (2) using a log2 scale*

*# obs1 <- data.frame(km = 2:7, time = (2:7)^2, group = c("A", "A", "A", "B", "B", "B")) ; obs1 ; fun\_gg\_scatter(data1 = obs1, x = "km", y = "time", x.log = "log2", x.tick.nb = 6)*

*### tick number. Example (3) using a log10 scale*

*# obs1 <- data.frame(km = 2:7, time = (2:7)^2, group = c("A", "A", "A", "B", "B", "B")) ; obs1 ; fun\_gg\_scatter(data1 = obs1, x = "km", y = "time", x.log = "log10", x.tick.nb = 6)*

*### tick number. Example (4) using a log10 scale: the reverse x-axis correctly deal with log10 scale*

*# obs1 <- data.frame(km = 2:7, time = (2:7)^2, group = c("A", "A", "A", "B", "B", "B")) ; obs1 ; fun\_gg\_scatter(data1 = obs1, x = "km", y = "time", x.log = "log10", x.lim = c(7, 2))*

*### secondary tick number. Example (1)*

*# obs1 <- data.frame(km = 2:7, time = (2:7)^2, group = c("A", "A", "A", "B", "B", "B")) ; obs1 ; fun\_gg\_scatter(data1 = obs1, x = "km", y = "time", x.inter.tick.nb = 4)*

*### secondary ticks. Example (2) not for log2 and log10 scales (see below warning message with the return argument)*

*# obs1 <- data.frame(km = 2:7, time = (2:7)^2, group = c("A", "A", "A", "B", "B", "B")) ; obs1 ; fun\_gg\_scatter(data1 = obs1, x = "km", y = "time", x.log = "log10", x.inter.tick.nb = 4)*

*### extra margins. To avoid dot cuts*

*# obs1 <- data.frame(km = 2:7, time = (2:7)^2, group = c("A", "A", "A", "B", "B", "B")) ; obs1 ; fun\_gg\_scatter(data1 = obs1, x = "km", y = "time", x.left.extra.margin = 0.25, x.right.extra.margin = 0.25)*

*### include zero in both the x-axis and y-xis*

*# obs1 <- data.frame(km = 2:7, time = (2:7)^2, group = c("A", "A", "A", "B", "B", "B")) ; obs1 ; fun\_gg\_scatter(data1 = obs1, x = "km", y = "time", xy.include.zero = TRUE)*

*### graph title, text size and legend display*

*# obs1 <- data.frame(km = 2:7, time = (2:7)^2, group = c("A", "A", "A", "B", "B", "B")) ; obs1 ; fun\_gg\_scatter(data1 = obs1, x = "km", y = "time", categ = "group", text.size = 8, title = "GRAPH1", title.text.size = 16, show.legend = TRUE)*

*### raster display. This switchs from vectorial mode to raster mode. The display can takes some time, but this is easier to export and handle than vectorial display*

*# set.seed(1) ; obs1 <- data.frame(km = rnorm(100000, 22, 3), time = rnorm(100000, 22, 3)) ; set.seed(NULL) ; fun\_gg\_scatter(data1 = obs1, x = "km", y = "time", raster = TRUE)*

*### classic representation (use grid = TRUE to display the background lines of the y axis ticks)*

*# obs1 <- data.frame(km = 2:7, time = (2:7)^2, group = c("A", "A", "A", "B", "B", "B")) ; obs1 ; fun\_gg\_scatter(data1 = obs1, x = "km", y = "time", classic = TRUE, grid = FALSE)*

*### graphic info. Example (1)*

*# obs1 <- data.frame(km = 2:7, time = (2:7)^2, group = c("A", "A", "A", "B", "B", "B")) ; fun\_gg\_scatter(data1 = obs1, x = "km", y = "time", return = TRUE)*

*### graphic info. Example (2) of assignation and warning message display*

*# obs1 <- data.frame(km = 2:7, time = (2:7)^2, group = c("A", "A", "A", "B", "B", "B")) ; output <- fun\_gg\_scatter(data1 = obs1, x = "km", y = "time", x.log = "log10", return = TRUE) ; cat(output$warn)*

*### add ggplot2 functions*

*# obs1 <- data.frame(km = 2:7, time = (2:7)^2, group = c("A", "A", "A", "B", "B", "B")) ; obs1 ; fun\_gg\_scatter(data1 = obs1, x = "km", y = "time", add = "+ggplot2::theme\_classic()")*

*### all the arguments*

*# set.seed(1) ; obs1 <- data.frame(km = rnorm(1000, 10, 3), time = rnorm(1000, 10, 3), group1 = rep(c("A1", "A2"), 500)) ; obs2 <-data.frame(km = rnorm(1000, 15, 3), time = rnorm(1000, 15, 3), group2 = rep(c("G1", "G2"), 500)) ; set.seed(NULL) ; obs1$L1$km[2:3] <- NA ; fun\_gg\_scatter(data1 = list(L1 = obs1, L2 = obs2), x = list(L1 = "km", L2 = "km"), y = list(L1 = "time", L2 = "time"), categ = list(L1 = "group1", L2 = "group2"), legend.name = NULL, color = list(L1 = 4:5, L2 = 7:8), geom = list(L1 = "geom\_point", L2 = "geom\_point"), alpha = list(L1 = 0.5, L2 = 0.5), dot.size = 3, line.size = 0.5, x.lim = c(1, 25), x.lab = "KM", x.log = "no", x.tick.nb = 10, x.inter.tick.nb = 1, x.left.extra.margin = 0, x.right.extra.margin = 0, y.lim = c(1, 25), y.lab = "TIME (s)", y.log = "log10", y.tick.nb = 5, y.inter.tick.nb = NULL, y.top.extra.margin = 0, y.bottom.extra.margin = 0, xy.include.zero = TRUE, text.size = 12, title = "", title.text.size = 8, show.legend = TRUE, classic = FALSE, grid = FALSE, raster = FALSE, vectorial.limit = NULL, return = FALSE, plot = TRUE, add = NULL, warn.print = TRUE, lib.path = NULL)*