######## fun\_gg\_scatter() #### ggplot2 scatterplot + lines (up to 6 overlays totally)

**# Check OK: clear to go Apollo**

**fun\_gg\_scatter <- function(data1, x, y, categ = NULL, legend.name = NULL, color = NULL, geom = "geom\_point", alpha = 0.5, dot.size = 2, line.size = 0.5, xlim = NULL, xlab = NULL, xlog = "no", x.tick.nb = NULL, x.inter.tick.nb = NULL, x.left.extra.margin = 0.05, x.right.extra.margin = 0.05, ylim = NULL, ylab = NULL, ylog = "no", y.tick.nb = NULL, y.inter.tick.nb = NULL, y.top.extra.margin = 0.05, y.bottom.extra.margin = 0.05, xy.include.zero = FALSE, text.size = 12, title = "", title.text.size = 12, show.legend = TRUE, classic = FALSE, grid = FALSE, raster = FALSE, vectorial.limit = NULL, return = FALSE, plot = TRUE, add = NULL, warn.print = FALSE, lib.path = NULL){**

*# 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 ggplot, or a list of data frames*

*# x: character string or list of character string (one compartment for each list compartment of data1) of the data1 column name for x-axis. write NULL for each "geom\_hline" in geom argument*

*# y: character string or list of character string (one compartment for each list compartment of data1) of the data1 column name for y-axis. Write NULL for each "geom\_vline" in geom argument*

*# categ: character string or list of character string (one compartment for each list compartment of data1) of the data1 column name for categories. If categ = NULL, no categories (no legend). Some of the list compartments can be NULL, and other not*

*# legend.name: character string list of character string (one compartment for each list compartment of data1) of the legend title. If legend.name = NULL and categ != NULL, then legend.name <- categ. Some of the list compartments can be NULL, and other not*

*# color: vector of character string or list of character vectors (one compartment for each list compartment of data1) for 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 color is a list, some of the compartments can be NULL. In that case, a different grey color will be used for each NULL compartment*

*# geom: character string or list of character string (one compartment for each list compartment of data1) for 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) xlim or ylim argument must NOT be NULL, respectively, if only these kind of lines are drawn (if other geom present, then xlim = NULL and ylim = NULL will generate xlim and ylim 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*

*# alpha: numeric value (from 0 to 1) of the transparency or list of numeric values (one compartment for each list compartment of data1)*

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

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

*# xlim: 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 xlim must be already in the corresponding log if xlog argument is not "no" (see below)*

*# xlab: 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*

*# xlog: 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: do not tranform the data, but just display ticks in a log scale manner. Thus, negative or zero values allowed. BEWARE: not possible to have horizontal bars with a log axis, due to a bug in ggplot2 (see https://github.com/tidyverse/ggplot2/issues/881)*

*# x.tick.nb: approximate number of desired label values on the x-axis (n argument of the the fun\_scale() function)*

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

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

*# ylim: 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 ylim must be already in the corresponding log if ylog argument is not "no" (see below)*

*# ylab: 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*

*# ylog: 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: do not tranform the data, but just display ticks in a log scale manner. Thus, negative or zero values allowed. BEWARE: not possible to have horizontal bars with a log axis, due to a bug in ggplot2 (see https://github.com/tidyverse/ggplot2/issues/881)*

*# y.tick.nb: approximate number of desired label values on the y-axis (n argument of the the fun\_scale() function)*

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

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

*# xy.include.zero: logical. Does xlim and ylim range include 0? Ok even if xlog = TRUE or ylog = TRUE because xlim and ylim must already be log transformed values*

*# 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*

*# classic: logical. Use the classic theme (article like)?*

*# 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*

*# 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*

*# warn.print: logical. Print warnings at the end of the execution? No print if no warning messages*

*# lib.path: absolute path of the required packages, if not in the 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, xlim = c(1, 25), xlab = "KM", xlog = "no", x.tick.nb = 10, x.inter.tick.nb = 1, x.left.extra.margin = 0, x.right.extra.margin = 0, ylim = c(1, 25), ylab = expression(paste("TIME (", 10^-20, " s)")), ylog = "log10", y.tick.nb = 5, y.top.extra.margin = 0, y.bottom.extra.margin = 0, xy.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 ggplot 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 (ggplot 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 ylab 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", xlim = 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 ylab 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", ylim = 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 ggplot 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), xlim = c(10, 40), ylim = 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 (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", xlim = c(-1, 25))*

*### x-axis limits. Example (2) showing that order matters in ylim 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", xlim = 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", xlog = "log10")*

*### log scale. Example (2). BEWARE: values of the xlim 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", xlog = "log10", xlim = 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", xlog = "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", xlog = "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", xlog = "log10", xlim = 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", xlog = "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", xlog = "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, xlim = c(1, 25), xlab = "KM", xlog = "no", x.tick.nb = 10, x.inter.tick.nb = 1, x.left.extra.margin = 0, x.right.extra.margin = 0, ylim = c(1, 25), ylab = "TIME (s)", ylog = "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)*

*# DEBUGGING*

*# 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 ; 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 ; xlim = c(25, 0) ; xlab = "KM" ; xlog = "no" ; x.tick.nb = 10 ; x.inter.tick.nb = 1 ; x.left.extra.margin = 0 ; x.right.extra.margin = 0 ; ylim = c(1, 25) ; ylab = "TIME (s)" ; ylog = "log2" ; y.tick.nb = 5 ; y.inter.tick.nb = 2 ; 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*

*# data1 <- list(L1 = data.frame(a = 1:6, b = (1:6)^2, group = c("A", "A", "A", "B", "B", "B")), L2 = data.frame(a = (1:6)\*2, b = ((1:6)^2)\*2, group = c("A1", "A1", "A1", "B1", "B1", "B1")), L3 = data.frame(a = (1:6)\*3, b = ((1:6)^2)\*3, group3 = c("A4", "A5", "A6", "A7", "B4", "B5"))) ; data1$L1$a[3] <- NA ; data1$L1$group[5] <- NA ; data1$L3$group3[4] <- NA ; x = list(L1 = names(data1$L1)[1], L2 = names(data1$L2)[1], L3 = NULL) ; y = list(L1 = names(data1$L1)[2], L2 = names(data1$L2)[2], L3 = "a") ; categ = list(L1 = "group", L2 = NULL, L3 = NULL) ; legend.name = NULL ; color = NULL ; geom = list(L1 = "geom\_point", L2 = "geom\_point", L3 = "geom\_hline") ; alpha = list(L1 = 0.5, L2 = 0.5, L3 = 0.5) ; dot.size = 1 ; line.size = 0.5 ; xlim = c(14, 4) ; xlab = NULL ; xlog = "log10" ; x.tick.nb = 10 ; x.inter.tick.nb = 4 ; x.left.extra.margin = 0 ; x.right.extra.margin = 0 ; ylim = c(60, 5) ; ylab = NULL ; ylog = "log10" ; y.tick.nb = 10 ; y.inter.tick.nb = 2 ; y.top.extra.margin = 0 ; y.bottom.extra.margin = 0 ; xy.include.zero = FALSE ; 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*

*# data1 <- data.frame(km = 1:2, time = (1:2)^2, group = c("A", "B")) ; data1 ; x = NULL; y = "km"; categ = "group"; legend.name = NULL ; color = NULL ; geom = "geom\_hline"; alpha = 0.5 ; dot.size = 1 ; line.size = 0.5 ; xlim = c(1,10) ; xlab = NULL ; xlog = "log10" ; x.tick.nb = 10 ; x.inter.tick.nb = 4 ; x.left.extra.margin = 0 ; x.right.extra.margin = 0 ; ylim = NULL ; ylab = expression(paste("TIME (", 10^-20, " s)")) ; ylog = "log10" ; y.tick.nb = 10 ; y.inter.tick.nb = 2 ; y.top.extra.margin = 0 ; y.bottom.extra.margin = 0 ; xy.include.zero = FALSE ; 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*

*# function name*

**function.name <- paste0(as.list(match.call(expand.dots=FALSE))[[1]], "()")**

*# end function name*

*# required function checking*

**req.function <- c(**

**"fun\_gg\_palette",**

**"fun\_gg\_point\_rast",**

**"fun\_pack",**

**"fun\_check",**

**)**

**for(i1 in req.function){**

**if(length(find(i1, mode = "function")) == 0){**

**tempo.cat <- paste0("\n\n================\n\nERROR IN ", function.name, ": REQUIRED ", i1, "() FUNCTION IS MISSING IN THE R ENVIRONMENT\n\n================\n\n")**

**stop(tempo.cat)**

**}**

**}**

*# end required function checking*

*# reserved words to avoid bugs (used in this function)*

**reserved.words <- c("fake\_x", "fake\_y", "fake\_categ", "color")**

*# end reserved words to avoid bugs (used in this function)*

*# primary argument checking*

**arg.check <- NULL** *#*

**text.check <- NULL** *#*

**checked.arg.names <- NULL** *# for function debbuging: used by r\_debugging\_tools*

**ee <- expression(arg.check <- c(arg.check, tempo$problem) , text.check <- c(text.check, tempo$text) , checked.arg.names <- c(checked.arg.names, tempo$fun.name))**

**tempo1 <- fun\_check(data = data1, class = "data.frame", na.contain = TRUE, fun.name = function.name, print = FALSE)**

**tempo2 <- fun\_check(data = data1, class = "list", na.contain = TRUE, fun.name = function.name, print = FALSE)**

**if(tempo1$problem == TRUE & tempo2$problem == TRUE){**

**tempo.cat <- paste0("ERROR IN ", function.name, ": data1 ARGUMENT MUST BE A DATA FRAME OR A LIST OF DATA FRAMES")**

**text.check <- c(text.check, tempo.cat)**

**arg.check <- c(arg.check, TRUE)**

**}**

**if(any(arg.check) == TRUE){**

**stop(paste0("\n\n================\n\n", paste(text.check[arg.check], collapse = "\n"), "\n\n================\n\n"), call. = FALSE)** *#*

**}**

*# source("C:/Users/Gael/Documents/Git\_versions\_to\_use/debugging\_tools\_for\_r\_dev-v1.2/r\_debugging\_tools-v1.2.R") ; eval(parse(text = str\_basic\_arg\_check\_dev)) ; eval(parse(text = str\_arg\_check\_with\_fun\_check\_dev)) # activate this line and use the function (with no arguments left as NULL) to check arguments status and if they have been checked using fun\_check()*

*# end primary argument checking*

*# second round of checking and data preparation*

*# end second round of checking and data preparation*

*# check list lengths (and names of data1 compartments if non name present)*

**warn <- NULL**

**warn.count <- 0**

**if(all(class(data1) == "list")){**

**if(length(data1) > 6){**

**tempo.cat <- paste0("\n\n================\n\nERROR IN ", function.name, ": data1 ARGUMENT MUST BE A LIST OF 6 DATA FRAMES MAXIMUM (6 OVERLAYS MAX)\n\n================\n\n")**

**stop(tempo.cat, call. = FALSE)**

**}**

**if(is.null(names(data1))){**

**names(data1) <- paste0("L", 1:length(data1))**

**warn.count <- warn.count + 1**

**tempo.warn <- paste0("(", warn.count,") FROM FUNCTION ", function.name, ": NULL NAME COMPARTMENT OF data1 LIST -> NAMES RESPECTIVELY ATTRIBUTED TO EACH COMPARTMENT:\n", paste(names(data1), collapse = " "))**

**warn <- paste0(ifelse(is.null(warn), tempo.warn, paste0(warn, "\n\n", tempo.warn)))**

**}**

**if( ! is.null(x)){**

**if( ! (all(class(x) == "list") & length(data1) == length(x))){**

**tempo.cat <- paste0("\n\n================\n\nERROR IN ", function.name, ": x ARGUMENT MUST BE A LIST OF SAME LENGTH AS data1 IF data1 IS A LIST\n\n================\n\n")**

**stop(tempo.cat, call. = FALSE)**

**}**

**}else{**

**x <- vector("list", length(data1))**

**}**

**if( ! is.null(y)){**

**if( ! (all(class(y) == "list") & length(data1) == length(y))){**

**tempo.cat <- paste0("\n\n================\n\nERROR IN ", function.name, ": y ARGUMENT MUST BE A LIST OF SAME LENGTH AS data1 IF data1 IS A LIST\n\n================\n\n")**

**stop(tempo.cat, call. = FALSE)**

**}**

**}else{**

**y <- vector("list", length(data1))**

**}**

**if( ! is.null(categ)){**

**if( ! (all(class(categ) == "list") & length(data1) == length(categ))){**

**tempo.cat <- paste0("\n\n================\n\nERROR IN ", function.name, ": categ ARGUMENT MUST BE A LIST OF SAME LENGTH AS data1 IF data1 IS A LIST\n\n================\n\n")**

**stop(tempo.cat, call. = FALSE)**

**}**

**}**

**if( ! is.null(legend.name)){**

**if( ! (all(class(legend.name) == "list") & length(data1) == length(legend.name))){**

**tempo.cat <- paste0("\n\n================\n\nERROR IN ", function.name, ": legend.name ARGUMENT MUST BE A LIST OF SAME LENGTH AS data1 IF data1 IS A LIST\n\n================\n\n")**

**stop(tempo.cat, call. = FALSE)**

**}**

**}**

**if( ! is.null(color)){**

**if( ! (all(class(color) == "list") & length(data1) == length(color))){**

**tempo.cat <- paste0("\n\n================\n\nERROR IN ", function.name, ": color ARGUMENT MUST BE A LIST OF SAME LENGTH AS data1 IF data1 IS A LIST\n\n================\n\n")**

**stop(tempo.cat, call. = FALSE)**

**}**

**}**

**if( ! (all(class(geom) == "list") & length(data1) == length(geom))){**

**tempo.cat <- paste0("\n\n================\n\nERROR IN ", function.name, ": geom ARGUMENT MUST BE A LIST OF SAME LENGTH AS data1 IF data1 IS A LIST\n\n================\n\n")**

**stop(tempo.cat, call. = FALSE)**

**}**

**if( ! (all(class(alpha) == "list") & length(data1) == length(alpha))){**

**tempo.cat <- paste0("\n\n================\n\nERROR IN ", function.name, ": alpha ARGUMENT MUST BE A LIST OF SAME LENGTH AS data1 IF data1 IS A LIST\n\n================\n\n")**

**stop(tempo.cat, call. = FALSE)**

**}**

**}**

*# end check list lengths (and names of data1 compartments if non name present)*

*# conversion into lists*

**if(all(is.data.frame(data1))){**

**data1 <- list(L1 = data1)**

**if(all(class(x) == "list")){**

**tempo.cat <- paste0("\n\n================\n\nERROR IN ", function.name, ": x ARGUMENT CANNOT BE A LIST IF data1 IS A DATA FRAME\n\n================\n\n")**

**stop(tempo.cat, call. = FALSE)**

**}else{**

**x <- list(L1 = x)**

**}**

**if(all(class(y) == "list")){**

**tempo.cat <- paste0("\n\n================\n\nERROR IN ", function.name, ": y ARGUMENT CANNOT BE A LIST IF data1 IS A DATA FRAME\n\n================\n\n")**

**stop(tempo.cat, call. = FALSE)**

**}else{**

**y <- list(L1 = y)**

**}**

**if( ! is.null(categ)){**

**if(all(class(categ) == "list")){**

**tempo.cat <- paste0("\n\n================\n\nERROR IN ", function.name, ": categ ARGUMENT CANNOT BE A LIST IF data1 IS A DATA FRAME\n\n================\n\n")**

**stop(tempo.cat, call. = FALSE)**

**}else{**

**categ <- list(L1 = categ)**

**}**

**}**

**if( ! is.null(legend.name)){**

**if(all(class(legend.name) == "list")){**

**tempo.cat <- paste0("\n\n================\n\nERROR IN ", function.name, ": legend.name ARGUMENT CANNOT BE A LIST IF data1 IS A DATA FRAME\n\n================\n\n")**

**stop(tempo.cat, call. = FALSE)**

**}else{**

**legend.name <- list(L1 = legend.name)**

**}**

**}**

**if( ! is.null(color)){**

**if(all(class(color) == "list")){**

**tempo.cat <- paste0("\n\n================\n\nERROR IN ", function.name, ": color ARGUMENT CANNOT BE A LIST IF data1 IS A DATA FRAME\n\n================\n\n")**

**stop(tempo.cat, call. = FALSE)**

**}else{**

**color <- list(L1 = color)**

**}**

**}**

**if(all(class(geom) == "list")){**

**tempo.cat <- paste0("\n\n================\n\nERROR IN ", function.name, ": geom ARGUMENT CANNOT BE A LIST IF data1 IS A DATA FRAME\n\n================\n\n")**

**stop(tempo.cat, call. = FALSE)**

**}else{**

**geom <- list(L1 = geom)**

**}**

**if(all(class(alpha) == "list")){**

**tempo.cat <- paste0("\n\n================\n\nERROR IN ", function.name, ": alpha ARGUMENT CANNOT BE A LIST IF data1 IS A DATA FRAME\n\n================\n\n")**

**stop(tempo.cat, call. = FALSE)**

**}else{**

**alpha <- list(L1 = alpha)**

**}**

**}**

*# end conversion into lists*

*# legend name filling*

**if(is.null(legend.name) & ! is.null(categ)){**

**legend.name <- categ**

**}else if(is.null(legend.name) & is.null(categ)){**

**legend.name <- vector("list", length(data1))** *# null list*

**}**

*# end legend name filling*

*# ini categ for legend display*

**fin.lg.disp <- vector("list", 6)** *# will be used at the end to display or not legends*

**fin.lg.disp[] <- FALSE**

**legend.disp <- vector("list", length(data1))**

**if(is.null(categ) | show.legend == FALSE){**

**legend.disp[] <- FALSE**

**}else{**

**for(i2 in 1:length(data1)){**

**if(is.null(categ[[i2]])){**

**legend.disp[[i2]] <- FALSE**

**}else{**

**legend.disp[[i2]] <- TRUE**

**}**

**}**

**}**

*# end ini categ for legend display*

*# integer colors into gg\_palette*

**tempo.check.color <- NULL**

**for(i1 in 1:length(data1)){**

**if(any(is.na(color[[i1]]))){**

**tempo.cat <- paste0("\n\n================\n\nERROR IN ", function.name, ": ", ifelse(length(color) == 1, "color", paste0("color NUMBER ", i1)), " IN ", ifelse(length(data1) == 1, "data1", paste0("data1 NUMBER ", i1)), ": color ARGUMENT CANNOT CONTAIN NA\n\n================\n\n")**

**stop(tempo.cat, call. = FALSE)**

**}**

**tempo.check.color <- c(tempo.check.color, fun\_check(data = color[[i1]], data.name = ifelse(length(color) == 1, "color", paste0("color NUMBER ", i1)), class = "integer", double.as.integer.allowed = TRUE, na.contain = TRUE, fun.name = function.name, print = FALSE)$problem)**

**}**

**tempo.check.color <- ! tempo.check.color** *# invert TRUE and FALSE because if integer, then problem = FALSE*

**if(any(tempo.check.color == TRUE)){** *# convert integers into colors*

**tempo.integer <- unlist(color[tempo.check.color])**

**tempo.color <- fun\_gg\_palette(max(tempo.integer, na.rm = TRUE))**

**for(i1 in 1:length(data1)){**

**if(tempo.check.color[i1] == TRUE){**

**color[[i1]] <-tempo.color[color[[i1]]]**

**}**

**}**

**}**

*# end integer colors into gg\_palette*

*# argument checking*

**arg.check <- NULL** *#*

**text.check <- NULL** *#*

**checked.arg.names <- NULL** *# for function debbuging: used by r\_debugging\_tools*

**ee <- expression(arg.check <- c(arg.check, tempo$problem) , text.check <- c(text.check, tempo$text) , checked.arg.names <- c(checked.arg.names, tempo$fun.name))**

**compart.null.color <- 0** *# will be used to attribute a color when color is non NULL but a compartment of color is NULL*

**data1.ini <- data1** *# to report NA removal*

**removed.row.nb <- vector("list", length = length(data1))** *# to report NA removal*

**removed.rows <- vector("list", length = length(data1))** *# to report NA removal*

**for(i1 in 1:length(data1)){**

**tempo <- fun\_check(data = data1[[i1]], data.name = ifelse(length(data1) == 1, "data1", paste0("data1 NUMBER ", i1)), class = "data.frame", na.contain = TRUE, fun.name = function.name) ; eval(ee)**

*# reserved word checking*

**if(any(names(data1[[i1]]) %in% reserved.words)){** *# I do not use fun\_name\_change() because cannot control y before creating "fake\_y". But ok because reserved are not that common*

**tempo.cat <- paste0("ERROR IN ", function.name, ": COLUMN NAMES OF ", ifelse(length(data1) == 1, "data1", paste0("data1 NUMBER ", i1)), " ARGUMENT CANNOT BE ONE OF THESE WORDS\n", paste(reserved.words, collapse = " "), "\nTHESE ARE RESERVED FOR THE ", function.name, " FUNCTION")**

**text.check <- c(text.check, tempo.cat)**

**arg.check <- c(arg.check, TRUE)**

**}**

*# end reserved word checking*

*# check of geom now because required for y argument*

**tempo <- fun\_check(data = geom[[i1]], data.name = ifelse(length(geom) == 1, "geom", paste0("geom NUMBER ", i1)), options = c("geom\_point", "geom\_line", "geom\_path", "geom\_hline", "geom\_vline"), length = 1, fun.name = function.name) ; eval(ee)**

*# end check of geom now because required for y argument*

**if(is.null(x[[i1]])){**

**if(all(geom[[i1]] != "geom\_hline")){**

**tempo.cat <- paste0("ERROR IN ", function.name, ": ", ifelse(length(x) == 1, "x", paste0("x NUMBER ", i1)), " IN ", ifelse(length(data1) == 1, "data1", paste0("data1 NUMBER ", i1)), ": x ARGUMENT CANNOT BE NULL EXCEPT IF ", ifelse(length(geom) == 1, "x", paste0("geom NUMBER ", i1)), " ARGUMENT IS \"geom\_hline\"\nHERE geom ARGUMENT IS: ", paste(geom[[i1]], collapse = " "))**

**text.check <- c(text.check, tempo.cat)**

**arg.check <- c(arg.check, TRUE)**

**}else{**

**x[[i1]] <- "fake\_x"**

**data1[[i1]] <- cbind(data1[[i1]], fake\_x = NA)**

**data1[[i1]][, "fake\_x"] <- as.numeric(data1[[i1]][, "fake\_x"])**

**warn.count <- warn.count + 1**

**tempo.warn <- paste0("(", warn.count,") FROM FUNCTION ", function.name, ": NULL ", ifelse(length(x) == 1, "x", paste0("x NUMBER ", i1)), " ARGUMENT ASSOCIATED TO ", ifelse(length(geom) == 1, "geom", paste0("geom NUMBER ", i1)), " ARGUMENT ", geom[[i1]], " -> FAKE COLUMN ADDED TO DATA FRAME ", ifelse(length(data1) == 1, "data1", paste0("data1 NUMBER ", i1)), ", NAMED \"fake\_x\" FOR FINAL DRAWING")**

**warn <- paste0(ifelse(is.null(warn), tempo.warn, paste0(warn, "\n\n", tempo.warn)))**

**}**

**}else{**

**if(all(geom[[i1]] == "geom\_hline")){**

**tempo.cat <- paste0("ERROR IN ", function.name, ": ", ifelse(length(x) == 1, "x", paste0("x NUMBER ", i1)), " IN ", ifelse(length(data1) == 1, "data1", paste0("data1 NUMBER ", i1)), ": x ARGUMENT MUST BE NULL IF ", ifelse(length(geom) == 1, "geom", paste0("geom NUMBER ", i1)), " ARGUMENT IS \"geom\_hline\"")**

**text.check <- c(text.check, tempo.cat)**

**arg.check <- c(arg.check, TRUE)**

**}**

**tempo <- fun\_check(data = x[[i1]], data.name = ifelse(length(x) == 1, "x", paste0("x NUMBER ", i1)), class = "vector", mode = "character", length = 1, fun.name = function.name) ; eval(ee)**

**}**

**if(is.null(y[[i1]])){**

**if(all(geom[[i1]] != "geom\_vline")){**

**tempo.cat <- paste0("ERROR IN ", function.name, ": ", ifelse(length(y) == 1, "y", paste0("y NUMBER ", i1)), " IN ", ifelse(length(data1) == 1, "data1", paste0("data1 NUMBER ", i1)), ": y ARGUMENT CANNOT BE NULL EXCEPT IF ", ifelse(length(geom) == 1, "y", paste0("geom NUMBER ", i1)), " ARGUMENT IS \"geom\_vline\"\nHERE geom ARGUMENT IS: ", paste(geom[[i1]], collapse = " "))**

**text.check <- c(text.check, tempo.cat)**

**arg.check <- c(arg.check, TRUE)**

**}else{**

**y[[i1]] <- "fake\_y"**

**data1[[i1]] <- cbind(data1[[i1]], fake\_y = NA)**

**data1[[i1]][, "fake\_y"] <- as.numeric(data1[[i1]][, "fake\_y"])**

**warn.count <- warn.count + 1**

**tempo.warn <- paste0("(", warn.count,") FROM FUNCTION ", function.name, ": NULL ", ifelse(length(y) == 1, "y", paste0("y NUMBER ", i1)), " ARGUMENT ASSOCIATED TO ", ifelse(length(geom) == 1, "geom", paste0("geom NUMBER ", i1)), " ARGUMENT ", geom[[i1]], " -> FAKE COLUMN ADDED TO DATA FRAME ", ifelse(length(data1) == 1, "data1", paste0("data1 NUMBER ", i1)), ", NAMED \"fake\_y\" FOR FINAL DRAWING")**

**warn <- paste0(ifelse(is.null(warn), tempo.warn, paste0(warn, "\n\n", tempo.warn)))**

**}**

**}else{**

**if(all(geom[[i1]] == "geom\_vline")){**

**tempo.cat <- paste0("ERROR IN ", function.name, ": ", ifelse(length(y) == 1, "y", paste0("y NUMBER ", i1)), " IN ", ifelse(length(data1) == 1, "data1", paste0("data1 NUMBER ", i1)), ": y ARGUMENT MUST BE NULL IF ", ifelse(length(geom) == 1, "geom", paste0("geom NUMBER ", i1)), " ARGUMENT IS \"geom\_vline\"")**

**text.check <- c(text.check, tempo.cat)**

**arg.check <- c(arg.check, TRUE)**

**}**

**tempo <- fun\_check(data = y[[i1]], data.name = ifelse(length(y) == 1, "y", paste0("y NUMBER ", i1)), class = "vector", mode = "character", length = 1, fun.name = function.name) ; eval(ee)**

**}**

**if( ! (x[[i1]] %in% names(data1[[i1]]))){**

**tempo.cat <- paste0("ERROR IN ", function.name, ": ", ifelse(length(x) == 1, "x", paste0("x NUMBER ", i1)), " ARGUMENT MUST BE A COLUMN NAME OF ", ifelse(length(data1) == 1, "data1", paste0("data1 NUMBER ", i1)))**

**text.check <- c(text.check, tempo.cat)**

**arg.check <- c(arg.check, TRUE)**

**}**

**if( ! (y[[i1]] %in% names(data1[[i1]]))){**

**tempo.cat <- paste0("ERROR IN ", function.name, ": ", ifelse(length(y) == 1, "y", paste0("y NUMBER ", i1)), " ARGUMENT MUST BE A COLUMN NAME OF ", ifelse(length(data1) == 1, "data1", paste0("data1 NUMBER ", i1)))**

**text.check <- c(text.check, tempo.cat)**

**arg.check <- c(arg.check, TRUE)**

**}**

*# na detection and removal (done now to be sure of the correct length of categ)*

**if(x[[i1]] == "fake\_x" & y[[i1]] == "fake\_y"){** *# because the code cannot accept to be both "fake\_x" and "fake\_y" at the same time*

**tempo.cat <- paste0("\n\n============\n\nERROR IN ", function.name, ": CODE INCONSISTENCY 2\nTHE CODE CANNOT ACCEPT x AND y TO BE \"fake\_x\" AND \"fake\_y\" IN THE SAME DATA FRAME ", i1, " \n\n============\n\n")**

**stop(tempo.cat, call. = FALSE)**

**}**

**if(any(is.na(data1[[i1]][, c(if(x[[i1]] == "fake\_x"){NULL}else{x[[i1]]}, if(y[[i1]] == "fake\_y"){NULL}else{y[[i1]]})]))){**

**tempo.removed.row.nb <- unlist(lapply(lapply(c(data1[[i1]][c(if(x[[i1]] == "fake\_x"){NULL}else{x[[i1]]}, if(y[[i1]] == "fake\_y"){NULL}else{y[[i1]]})]), FUN = is.na), FUN = which))**

**removed.row.nb[[i1]] <- c(removed.row.nb[[i1]], tempo.removed.row.nb)**

*# report of removed rows will be performed at the very end*

**data1[[i1]] <- data1[[i1]][-tempo.removed.row.nb, ]**

**warn.count <- warn.count + 1**

**tempo.warn <- paste0("(", warn.count,") FROM FUNCTION ", function.name, ": NA DETECTED IN COLUMN ", if(x[[i1]] == "fake\_x"){""}else{ifelse(length(x) == 1, "x", paste0("x NUMBER ", i1))}, if(x[[i1]] != "fake\_x" & y[[i1]] != "fake\_y"){" AND "}, if(y[[i1]] == "fake\_y"){""}else{ifelse(length(y) == 1, "y", paste0("y NUMBER ", i1))}, " IN ", ifelse(length(data1) == 1, "data1", paste0("data1 NUMBER ", i1)), ". CORRESPONDING ROWS HAVE BEEN REMOVED (SEE $removed.row.nb AND $removed.rows)")**

**warn <- paste0(ifelse(is.null(warn), tempo.warn, paste0(warn, "\n\n", tempo.warn)))**

**}**

*# end na detection and removal (done now to be sure of the correct length of categ)*

**tempo <- fun\_check(data = data1[[i1]][, x[[i1]]], data.name = ifelse(length(x) == 1, "x OF data1", paste0("x NUMBER ", i1, " OF data1 NUMBER ", i1)), class = "vector", mode = "numeric", na.contain = ifelse(x[[i1]] == "fake\_x", TRUE, FALSE), fun.name = function.name) ; eval(ee)**

**tempo <- fun\_check(data = data1[[i1]][, y[[i1]]], data.name = ifelse(length(y) == 1, "y OF data1", paste0("y NUMBER ", i1, " OF data1 NUMBER ", i1)), class = "vector", mode = "numeric", na.contain = ifelse(y[[i1]] == "fake\_y", TRUE, FALSE), fun.name = function.name) ; eval(ee)**

**if(( ! is.null(categ)) & ( ! is.null(categ[[i1]]))){** *# if categ[[i1]] = NULL, fake\_categ will be created later on*

**tempo <- fun\_check(data = categ[[i1]], data.name = ifelse(length(categ) == 1, "categ", paste0("categ NUMBER ", i1)),, class = "vector", mode = "character", length = 1, fun.name = function.name)**

**if( ! (categ[[i1]] %in% names(data1[[i1]]))){**

**tempo.cat <- paste0("ERROR IN ", function.name, ": ", ifelse(length(categ) == 1, "categ", paste0("categ NUMBER ", i1)), " ARGUMENT MUST BE A COLUMN NAME OF ", ifelse(length(data1) == 1, "data1", paste0("data1 NUMBER ", i1)))**

**text.check <- c(text.check, tempo.cat)**

**arg.check <- c(arg.check, TRUE)**

**}**

*# na detection and removal (done now to be sure of the correct length of categ)*

**if(any(is.na(data1[[i1]][, categ[[i1]]]))){**

**tempo.removed.row.nb <- unlist(lapply(lapply(c(data1[[i1]][categ[[i1]]]), FUN = is.na), FUN = which))**

**removed.row.nb[[i1]] <- c(removed.row.nb[[i1]], tempo.removed.row.nb)**

*# report of removed rows will be performed at the very end*

**data1[[i1]] <- data1[[i1]][-tempo.removed.row.nb, ]**

**warn.count <- warn.count + 1**

**tempo.warn <- paste0("(", warn.count,") FROM FUNCTION ", function.name, ": IN ", ifelse(length(categ) == 1, "categ", paste0("categ NUMBER ", i1)), " IN ", ifelse(length(data1) == 1, "data1", paste0("data1 NUMBER ", i1)), ", THE CATEGORY COLUMN:\n", paste(categ[[i1]], collapse = " "), "\nCONTAINS NA")**

**warn <- paste0(ifelse(is.null(warn), tempo.warn, paste0(warn, "\n\n", tempo.warn)))**

**}**

*# end na detection and removal (done now to be sure of the correct length of categ)*

**tempo1 <- fun\_check(data = data1[[i1]][, categ[[i1]]], data.name = ifelse(length(categ) == 1, "categ OF data1", paste0("categ NUMBER ", i1, " OF data1 NUMBER ", i1)), class = "vector", mode = "character", na.contain = FALSE, fun.name = function.name, print = FALSE)**

**tempo2 <- fun\_check(data = data1[[i1]][, categ[[i1]]], data.name = ifelse(length(categ) == 1, "categ OF data1", paste0("categ NUMBER ", i1, " OF data1 NUMBER ", i1)), class = "factor", na.contain = FALSE, fun.name = function.name, print = FALSE)**

**if(tempo1$problem == TRUE & tempo2$problem == TRUE){**

**tempo.cat <- paste0("ERROR IN ", function.name, ": ", ifelse(length(categ) == 1, "categ OF data1", paste0("categ NUMBER ", i1, " OF data1 NUMBER ", i1)), " MUST BE A FACTOR OR CHARACTER VECTOR")**

**text.check <- c(text.check, tempo.cat)**

**arg.check <- c(arg.check, TRUE)**

**}else if(tempo1$problem == FALSE){**

**data1[[i1]][, categ[[i1]]] <- factor(data1[[i1]][, categ[[i1]]])** *# if already a factor, change nothing, if characters, levels according to alphabetical order*

**warn.count <- warn.count + 1**

**tempo.warn <- paste0("(", warn.count,") FROM FUNCTION ", function.name, ": IN ", ifelse(length(categ) == 1, "categ", paste0("categ NUMBER ", i1)), " IN ", ifelse(length(data1) == 1, "data1", paste0("data1 NUMBER ", i1)), ", THE CHARACTER COLUMN HAS BEEN CONVERTED TO FACTOR")**

**warn <- paste0(ifelse(is.null(warn), tempo.warn, paste0(warn, "\n\n", tempo.warn)))**

**}**

**if(geom[[i1]] == "geom\_vline" | geom[[i1]] == "geom\_hline"){**

**if(length(unique(data1[[i1]][, categ[[i1]]])) != nrow(data1[[i1]])){**

**tempo.cat <- paste0("ERROR IN ", function.name, ": ", ifelse(length(geom) == 1, "geom OF data1", paste0("geom NUMBER ", i1, " OF data1 NUMBER ", i1)), " ARGUMENT IS ", geom[[i1]], ", MEANING THAT ", ifelse(length(categ) == 1, "categ OF data1", paste0("categ NUMBER ", i1, " OF data1 NUMBER ", i1)), " MUST HAVE A DIFFERENT CLASS PER LINE OF data1 (ONE x VALUE PER CLASS)")**

**text.check <- c(text.check, tempo.cat)**

**arg.check <- c(arg.check, TRUE)**

**}**

**}**

**}else if(( ! is.null(categ)) & is.null(categ[[i1]])){** *# if categ[[i1]] = NULL, fake\_categ will be created. BEWARE: is.null(categ[[i1]]) means no legend display (see above), because categ has not been precised. This also means a single color for data1[[i1]]*

**if(length(color[[i1]]) > 1){** *# 0 means is.null(color[[i1]]) and 1 is ok -> single color for data1[[i1]]*

**warn.count <- warn.count + 1**

**tempo.warn <- paste0("(", warn.count,") FROM FUNCTION ", function.name, ": NULL ", ifelse(length(categ) == 1, "categ", paste0("categ NUMBER ", i1)), " ARGUMENT BUT CORRESPONDING COLORS IN ", ifelse(length(color) == 1, "color", paste0("color NUMBER ", i1)), " HAS LENGTH OVER 1\n", paste(color[[i1]], collapse = " "), "\nWHICH IS NOT COMPATIBLE WITH NULL CATEG -> COLOR RESET TO A SINGLE COLOR")**

**warn <- paste0(ifelse(is.null(warn), tempo.warn, paste0(warn, "\n\n", tempo.warn)))**

**color[[i1]] <- NULL** *# will provide a single color below*

**}**

**categ[[i1]] <- "fake\_categ"**

**data1[[i1]] <- cbind(data1[[i1]], fake\_categ = "")**

*# inactivated because give a different color to different "Line\_" categ while a single color for all the data1[[i1]] required. Thus, put back after the color management*

*# if(geom[[i1]] == "geom\_hline" | geom[[i1]] == "geom\_vline"){*

*# data1[[i1]][, "fake\_categ"] <- paste0("Line\_", 1:nrow(data1[[i1]]))*

*# }else{*

**data1[[i1]][, "fake\_categ"] <- data1[[i1]][, "fake\_categ"]** *# as.numeric("") create a vector of NA but class numeric*

*# }*

**warn.count <- warn.count + 1**

**tempo.warn <- paste0("(", warn.count,") FROM FUNCTION ", function.name, ": NULL ", ifelse(length(categ) == 1, "categ", paste0("categ NUMBER ", i1)), " ARGUMENT -> FAKE COLUMN ADDED TO DATA FRAME ", ifelse(length(data1) == 1, "data1", paste0("data1 NUMBER ", i1)), ", NAMED \"fake\_categ\" FOR FINAL DRAWING")**

**warn <- paste0(ifelse(is.null(warn), tempo.warn, paste0(warn, "\n\n", tempo.warn)))**

**}**

**if( ! is.null(legend.name[[i1]])){**

**tempo <- fun\_check(data = legend.name[[i1]], data.name = ifelse(length(legend.name) == 1, "legend.name", paste0("legend.name NUMBER ", i1)),, class = "vector", mode = "character", length = 1, fun.name = function.name)**

**}**

**if( ! is.null(color)){** *# if color is NULL, will be filled later on*

*# check the nature of color*

**if(is.null(color[[i1]])){**

**compart.null.color <- compart.null.color + 1**

**color[[i1]] <- grey(compart.null.color / 8)** *# cannot be more than 7 overlays. Thus 7 different greys. 8/8 is excluded because white dots*

**warn.count <- warn.count + 1**

**tempo.warn <- paste0("(", warn.count,") FROM FUNCTION ", function.name, ": NULL COLOR IN ", ifelse(length(color) == 1, "color", paste0("color NUMBER ", i1)), " IN ", ifelse(length(data1) == 1, "data1", paste0("data1 NUMBER ", i1)), ", SINGLE COLOR ", paste(color[[i1]], collapse = " "), " HAS BEEN ATTRIBUTED")**

**warn <- paste0(ifelse(is.null(warn), tempo.warn, paste0(warn, "\n\n", tempo.warn)))**

**}**

**tempo1 <- fun\_check(data = color[[i1]], data.name = ifelse(length(color) == 1, "color", paste0("color NUMBER ", i1)), class = "vector", mode = "character", na.contain = TRUE, fun.name = function.name, print = FALSE)**

**tempo2 <- fun\_check(data = color[[i1]], data.name = ifelse(length(color) == 1, "color", paste0("color NUMBER ", i1)), class = "factor", na.contain = TRUE, fun.name = function.name, print = FALSE)**

**if(tempo1$problem == TRUE & tempo2$problem == TRUE){**

**tempo.cat <- paste0("ERROR IN ", function.name, ": ", ifelse(length(color) == 1, "color", paste0("color NUMBER ", i1)), " MUST BE A FACTOR OR CHARACTER VECTOR OR INTEGER VECTOR")** *# integer possible because dealt above*

**text.check <- c(text.check, tempo.cat)**

**arg.check <- c(arg.check, TRUE)**

**}else if( ! (all(color[[i1]] %in% colors() | grepl(pattern = "^#", color[[i1]])))){** *# check that all strings of low.color start by #*

**tempo.cat <- paste0("ERROR IN ", function.name, ": ", ifelse(length(color) == 1, "color", paste0("color NUMBER ", i1)), " ARGUMENT MUST BE A HEXADECIMAL COLOR VECTOR STARTING BY # AND/OR COLOR NAMES GIVEN BY colors(): ", paste(unique(color[[i1]]), collapse = " "))**

**text.check <- c(text.check, tempo.cat)**

**arg.check <- c(arg.check, TRUE)**

**}**

**if(any(is.na(color[[i1]]))){**

**warn.count <- warn.count + 1**

**tempo.warn <- paste0("(", warn.count,") FROM FUNCTION ", function.name, ": IN ", ifelse(length(color) == 1, "color", paste0("color NUMBER ", i1)), " IN ", ifelse(length(data1) == 1, "data1", paste0("data1 NUMBER ", i1)), ", THE COLORS:\n", paste(unique(color[[i1]]), collapse = " "), "\nCONTAINS NA")**

**warn <- paste0(ifelse(is.null(warn), tempo.warn, paste0(warn, "\n\n", tempo.warn)))**

**}**

*# end check the nature of color*

*# check the length of color*

**if(is.null(categ) & length(color[[i1]]) != 1){**

**tempo.cat <- paste0("ERROR IN ", function.name, ": ", ifelse(length(color) == 1, "color", paste0("color NUMBER ", i1)), " ARGUMENT MUST BE A SINGLE COLOR IF categ IS NULL")**

**text.check <- c(text.check, tempo.cat)**

**arg.check <- c(arg.check, TRUE)**

**}else if( ! is.null(categ)){**

*# No problem of NA management by ggplot2 because already removed*

**if(categ[[i1]] == "fake\_categ" & length(color[[i1]]) != 1){**

**tempo.cat <- paste0("ERROR IN ", function.name, ": ", ifelse(length(color) == 1, "color", paste0("color NUMBER ", i1)), " ARGUMENT MUST BE A SINGLE COLOR IF ", ifelse(length(categ) == 1, "categ", paste0("categ NUMBER ", i1)), " IS NULL")**

**text.check <- c(text.check, tempo.cat)**

**arg.check <- c(arg.check, TRUE)**

**}else if(length(color[[i1]]) == length(unique(data1[[i1]][, categ[[i1]]]))){** *# here length(color) is equal to the different number of categ*

**data1[[i1]][, categ[[i1]]] <- factor(data1[[i1]][, categ[[i1]]])** *# if already a factor, change nothing, if characters, levels according to alphabetical order*

**warn.count <- warn.count + 1**

**tempo.warn <- paste0("(", warn.count,") FROM FUNCTION ", function.name, ": IN ", ifelse(length(categ) == 1, "categ", paste0("categ NUMBER ", i1)), " IN ", ifelse(length(data1) == 1, "data1", paste0("data1 NUMBER ", i1)), ", THE FOLLOWING COLORS:\n", paste(color[[i1]], collapse = " "), "\nHAVE BEEN ATTRIBUTED TO THESE CLASSES:\n", paste(levels(factor(data1[[i1]][, categ[[i1]]])), collapse = " "))**

**warn <- paste0(ifelse(is.null(warn), tempo.warn, paste0(warn, "\n\n", tempo.warn)))**

**}else if(length(color[[i1]]) == length(data1[[i1]][, categ[[i1]]])){***# here length(color) is equal to nrow(data1[[i1]]) -> Modif to have length(color) equal to the different number of categ (length(color) == length(levels(data1[[i1]][, categ[[i1]]])))*

**data1[[i1]] <- cbind(data1[[i1]], color = color[[i1]])**

**tempo.check <- unique(data1[[i1]][ , c(categ[[i1]], "color")])**

**if( ! (nrow(data1[[i1]]) == length(color[[i1]]) & nrow(tempo.check) == length(unique(data1[[i1]][ , categ[[i1]]])))){**

**tempo.cat <- paste0("ERROR IN ", function.name, ": ", ifelse(length(color) == 1, "color", paste0("color NUMBER ", i1)), " ARGUMENT HAS THE LENGTH OF ", ifelse(length(categ) == 1, "categ", paste0("categ NUMBER ", i1)), " IN ", ifelse(length(data1) == 1, "data1", paste0("data1 NUMBER ", i1)), " COLUMN VALUES\nBUT IS INCORRECTLY ASSOCIATED TO EACH CLASS OF THIS categ:\n", paste(unique(mapply(FUN = "paste", data1[[i1]][ ,categ[[i1]]], data1[[i1]][ ,"color"])), collapse = "\n"))**

**text.check <- c(text.check, tempo.cat)**

**arg.check <- c(arg.check, TRUE)**

**}else{**

**data1[[i1]][, categ[[i1]]] <- factor(data1[[i1]][, categ[[i1]]])** *# if already a factor, change nothing, if characters, levels according to alphabetical order*

**color[[i1]] <- unique(color[[i1]][order(data1[[i1]][, categ[[i1]]])])** *# Modif to have length(color) equal to the different number of categ (length(color) == length(levels(data1[[i1]][, categ[[i1]]])))*

**warn.count <- warn.count + 1**

**tempo.warn <- paste0("(", warn.count, ") FROM FUNCTION ", function.name, ": ", ifelse(length(color) == 1, "color", paste0("color NUMBER ", i1)), " ARGUMENT HAS THE LENGTH OF ", ifelse(length(categ) == 1, "categ", paste0("categ NUMBER ", i1)), " IN ", ifelse(length(data1) == 1, "data1", paste0("data1 NUMBER ", i1)), " COLUMN VALUES\nCOLORS HAVE BEEN RESPECTIVELY ASSOCIATED TO EACH CLASS OF categ AS:\n", paste(levels(factor(data1[[i1]][, categ[[i1]]])), collapse = " "), "\n", paste(color[[i1]], collapse = " "))**

**warn <- paste0(ifelse(is.null(warn), tempo.warn, paste0(warn, "\n\n", tempo.warn)))**

**}**

**}else if(length(color[[i1]]) == 1){**

**data1[[i1]][, categ[[i1]]] <- factor(data1[[i1]][, categ[[i1]]])** *# if already a factor, change nothing, if characters, levels according to alphabetical order*

**color[[i1]] <- rep(color[[i1]], length(levels(data1[[i1]][, categ[[i1]]])))**

**warn.count <- warn.count + 1**

**tempo.warn <- paste0("(", warn.count,") FROM FUNCTION ", function.name, ": IN ", ifelse(length(categ) == 1, "categ", paste0("categ NUMBER ", i1)), " IN ", ifelse(length(data1) == 1, "data1", paste0("data1 NUMBER ", i1)), ", COLOR HAS LENGTH 1 MEANING THAT ALL THE DIFFERENT CLASSES OF ", ifelse(length(categ) == 1, "categ", paste0("categ NUMBER ", i1)), "\n", paste(levels(factor(data1[[i1]][, categ[[i1]]])), collapse = " "), "\nWILL HAVE THE SAME COLOR\n", paste(color[[i1]], collapse = " "))**

**warn <- paste0(ifelse(is.null(warn), tempo.warn, paste0(warn, "\n\n", tempo.warn)))**

**}else{**

**tempo.cat <- paste0("ERROR IN ", function.name, ": ", ifelse(length(color) == 1, "color", paste0("color NUMBER ", i1)), " ARGUMENT MUST BE (1) LENGTH 1, OR (2) THE LENGTH OF ", ifelse(length(categ) == 1, "categ", paste0("categ NUMBER ", i1)), " IN ", ifelse(length(data1) == 1, "data1", paste0("data1 NUMBER ", i1)), " COLUMN VALUES, OR (3) THE LENGTH OF THE CLASSES IN THIS COLUMN. HERE IT IS COLOR LENGTH ", length(color[[i1]]), " VERSUS CATEG LENGTH ", length(data1[[i1]][, categ[[i1]]]), " AND CATEG CLASS LENGTH ", length(unique(data1[[i1]][, categ[[i1]]])))**

**text.check <- c(text.check, tempo.cat)**

**arg.check <- c(arg.check, TRUE)**

**}**

**}**

**}**

**if((geom[[i1]] == "geom\_hline" | geom[[i1]] == "geom\_vline") & ! is.null(categ[[i1]])){** *# add here after the color management, to deal with the different lines to plot inside any data[[i1]]*

**if(categ[[i1]] == "fake\_categ"){**

**data1[[i1]][, "fake\_categ"] <- paste0("Line\_", 1:nrow(data1[[i1]]))**

**}**

**}**

**tempo <- fun\_check(data = alpha[[i1]], data.name = ifelse(length(color) == 1, "color", paste0("color NUMBER ", i1)), prop = TRUE, length = 1, fun.name = function.name) ; eval(ee)**

**}**

**if(length(data1) > 1){**

**if(length(unique(unlist(x))) > 1){**

**warn.count <- warn.count + 1**

**tempo.warn <- paste0("(", warn.count,") FROM FUNCTION ", function.name, ": THE x ARGUMENT DOES NOT CONTAIN IDENTICAL COLUMN NAMES:\n", paste(unlist(x), collapse = " "), "\nX-AXIS OVERLAYING DIFFERENT VARIABLES?")**

**warn <- paste0(ifelse(is.null(warn), tempo.warn, paste0(warn, "\n\n", tempo.warn)))**

**}**

**}**

**if(length(data1) > 1){**

**if(length(unique(unlist(y))) > 1){**

**warn.count <- warn.count + 1**

**tempo.warn <- paste0("(", warn.count,") FROM FUNCTION ", function.name, ": THE y ARGUMENT DOES NOT CONTAIN IDENTICAL COLUMN NAMES:\n", paste(unlist(y), collapse = " "), "\nY-AXIS OVERLAYING DIFFERENT VARIABLES?")**

**warn <- paste0(ifelse(is.null(warn), tempo.warn, paste0(warn, "\n\n", tempo.warn)))**

**}**

**}**

**if(sum(geom %in% "geom\_point") > 3){**

**tempo.cat <- paste0("ERROR IN ", function.name, ": geom ARGUMENT CANNOT HAVE MORE THAN THREE \"geom\_point\" ELEMENTS")**

**text.check <- c(text.check, tempo.cat)**

**arg.check <- c(arg.check, TRUE)**

**}else if(length(geom) - sum(geom %in% "geom\_point") > 3){**

**tempo.cat <- paste0("ERROR IN ", function.name, ": geom ARGUMENT CANNOT HAVE MORE THAN THREE LINE ELEMENTS")**

**text.check <- c(text.check, tempo.cat)**

**arg.check <- c(arg.check, TRUE)**

**}**

**tempo <- fun\_check(data = dot.size, class = "vector", mode = "numeric", length = 1, neg.values = FALSE, fun.name = function.name) ; eval(ee)**

**tempo <- fun\_check(data = line.size, class = "vector", mode = "numeric", length = 1, neg.values = FALSE, fun.name = function.name) ; eval(ee)**

**if( ! is.null(xlim)){**

**tempo <- fun\_check(data = xlim, class = "vector", mode = "numeric", length = 2, fun.name = function.name) ; eval(ee)**

**if(tempo$problem == FALSE & any(xlim %in% c(Inf, -Inf))){**

**tempo.cat <- paste0("ERROR IN ", function.name, ": xlim ARGUMENT CANNOT CONTAIN -Inf OR Inf VALUES")**

**text.check <- c(text.check, tempo.cat)**

**arg.check <- c(arg.check, TRUE)**

**}**

**}**

**if( ! is.null(xlab)){**

**if(all(class(xlab) %in% "expression")){** *# to deal with math symbols*

**tempo <- fun\_check(data = xlab, class = "expression", length = 1, fun.name = function.name) ; eval(ee)**

**}else{**

**tempo <- fun\_check(data = xlab, class = "vector", mode = "character", length = 1, fun.name = function.name) ; eval(ee)**

**}**

**}**

**tempo <- fun\_check(data = xlog, options = c("no", "log2", "log10"), length = 1, fun.name = function.name) ; eval(ee)**

**if(tempo$problem == FALSE & xlog != "no"){**

**warn.count <- warn.count + 1**

**tempo.warn <- paste0("(", warn.count,") FROM FUNCTION ", function.name, ": xlog ARGUMENT SET TO ", xlog, ".\nVALUES FROM THE x ARGUMENT COLUMN OF THE data1 DATA FRAME MUST BE ALREADY ", toupper(xlog), " TRANSFORMED, AS THE xlog ARGUMENT JUST MODIFIES THE AXIS SCALE")**

**warn <- paste0(ifelse(is.null(warn), tempo.warn, paste0(warn, "\n\n", tempo.warn)))**

**if( ! is.null(xlim)){**

**if(any(xlim <= 0)){**

**warn.count <- warn.count + 1**

**tempo.warn <- paste0("(", warn.count,") FROM FUNCTION ", function.name, ": xlim ARGUMENT CAN SPAN ZERO OR NEGATIVE VALUES IF xlog ARGUMENT IS SET TO ", xlog, " BECAUSE THIS LATTER ARGUMENT DOES NOT TRANSFORM DATA, JUST MODIFIES THE AXIS SCALE")**

**warn <- paste0(ifelse(is.null(warn), tempo.warn, paste0(warn, "\n\n", tempo.warn)))**

**}else if(any( ! is.finite(if(xlog == "log10"){10^xlim}else{2^xlim}))){**

**tempo.cat <- paste0("ERROR IN ", function.name, ": xlim ARGUMENT RETURNS INF WITH THE xlog ARGUMENT SET TO ", xlog, "\nAS SCALE COMPUTATION IS ", ifelse(xlog == "log10", 10, 2), "^xlim:\n", paste(ifelse(xlog == "log10", 10, 2)^xlim, collapse = " "), "\nARE YOU SURE THAT xlim ARGUMENT HAS BEEN SPECIFIED WITH VALUES ALREADY IN LOG SCALE?\n", paste(xlim, collapse = " "))**

**text.check <- c(text.check, tempo.cat)**

**arg.check <- c(arg.check, TRUE)**

**}**

**}**

**}**

**if( ! is.null(x.tick.nb)){**

**tempo <- fun\_check(data = x.tick.nb, class = "vector", typeof = "integer", length = 1, double.as.integer.allowed = TRUE, fun.name = function.name) ; eval(ee)**

**if(tempo$problem == FALSE & x.tick.nb < 0){**

**tempo.cat <- paste0("ERROR IN ", function.name, ": x.tick.nb ARGUMENT MUST BE A NON NULL POSITIVE INTEGER")**

**text.check <- c(text.check, tempo.cat)**

**arg.check <- c(arg.check, TRUE)**

**}**

**}**

**if( ! is.null(x.inter.tick.nb)){**

**tempo <- fun\_check(data = x.inter.tick.nb, class = "vector", typeof = "integer", length = 1, double.as.integer.allowed = TRUE, fun.name = function.name) ; eval(ee)**

**if(tempo$problem == FALSE & x.inter.tick.nb < 0){**

**tempo.cat <- paste0("ERROR IN ", function.name, ": x.inter.tick.nb ARGUMENT MUST BE A NON NULL POSITIVE INTEGER")**

**text.check <- c(text.check, tempo.cat)**

**arg.check <- c(arg.check, TRUE)**

**}**

**}**

**tempo <- fun\_check(data = x.left.extra.margin, prop = TRUE, length = 1, fun.name = function.name) ; eval(ee)**

**tempo <- fun\_check(data = x.right.extra.margin, prop = TRUE, length = 1, fun.name = function.name) ; eval(ee)**

**if( ! is.null(ylim)){**

**tempo <- fun\_check(data = ylim, class = "vector", mode = "numeric", length = 2, fun.name = function.name) ; eval(ee)**

**if(tempo$problem == FALSE & any(ylim %in% c(Inf, -Inf))){**

**tempo.cat <- paste0("ERROR IN ", function.name, ": ylim ARGUMENT CANNOT CONTAIN -Inf OR Inf VALUES")**

**text.check <- c(text.check, tempo.cat)**

**arg.check <- c(arg.check, TRUE)**

**}**

**}**

**if( ! is.null(ylab)){**

**if(all(class(ylab) %in% "expression")){** *# to deal with math symbols*

**tempo <- fun\_check(data = ylab, class = "expression", length = 1, fun.name = function.name) ; eval(ee)**

**}else{**

**tempo <- fun\_check(data = ylab, class = "vector", mode = "character", length = 1, fun.name = function.name) ; eval(ee)**

**}**

**}**

**tempo <- fun\_check(data = ylog, options = c("no", "log2", "log10"), length = 1, fun.name = function.name) ; eval(ee)**

**if(tempo$problem == FALSE & ylog != "no"){**

**warn.count <- warn.count + 1**

**tempo.warn <- paste0("(", warn.count,") FROM FUNCTION ", function.name, ": ylog ARGUMENT SET TO ", ylog, ".\nVALUES FROM THE y ARGUMENT COLUMN OF THE data1 DATA FRAME MUST BE ALREADY ", toupper(ylog), " TRANSFORMED, AS THE ylog ARGUMENT JUST MODIFIES THE AXIS SCALE")**

**warn <- paste0(ifelse(is.null(warn), tempo.warn, paste0(warn, "\n\n", tempo.warn)))**

**if( ! is.null(ylim)){**

**if(any(ylim <= 0)){**

**warn.count <- warn.count + 1**

**tempo.warn <- paste0("(", warn.count,") FROM FUNCTION ", function.name, ": ylim ARGUMENT CAN SPAN ZERO OR NEGATIVE VALUES IF ylog ARGUMENT IS SET TO ", ylog, " BECAUSE THIS LATTER ARGUMENT DOES NOT TRANSFORM DATA, JUST MODIFIES THE AXIS SCALE")**

**warn <- paste0(ifelse(is.null(warn), tempo.warn, paste0(warn, "\n\n", tempo.warn)))**

**}else if(any( ! is.finite(if(ylog == "log10"){10^ylim}else{2^ylim}))){**

**tempo.cat <- paste0("ERROR IN ", function.name, ": ylim ARGUMENT RETURNS INF WITH THE ylog ARGUMENT SET TO ", ylog, "\nAS SCALE COMPUTATION IS ", ifelse(ylog == "log10", 10, 2), "^ylim:\n", paste(ifelse(ylog == "log10", 10, 2)^ylim, collapse = " "), "\nARE YOU SURE THAT ylim ARGUMENT HAS BEEN SPECIFIED WITH VALUES ALREADY IN LOG SCALE?\n", paste(ylim, collapse = " "))**

**text.check <- c(text.check, tempo.cat)**

**arg.check <- c(arg.check, TRUE)**

**}**

**}**

**}**

**if( ! is.null(y.tick.nb)){**

**tempo <- fun\_check(data = y.tick.nb, class = "vector", typeof = "integer", length = 1, double.as.integer.allowed = TRUE, fun.name = function.name) ; eval(ee)**

**if(tempo$problem == FALSE & y.tick.nb < 0){**

**tempo.cat <- paste0("ERROR IN ", function.name, ": y.tick.nb ARGUMENT MUST BE A NON NULL POSITIVE INTEGER")**

**text.check <- c(text.check, tempo.cat)**

**arg.check <- c(arg.check, TRUE)**

**}**

**}**

**if( ! is.null(y.inter.tick.nb)){**

**tempo <- fun\_check(data = y.inter.tick.nb, class = "vector", typeof = "integer", length = 1, double.as.integer.allowed = TRUE, fun.name = function.name) ; eval(ee)**

**if(tempo$problem == FALSE & y.inter.tick.nb < 0){**

**tempo.cat <- paste0("ERROR IN ", function.name, ": y.inter.tick.nb ARGUMENT MUST BE A NON NULL POSITIVE INTEGER")**

**text.check <- c(text.check, tempo.cat)**

**arg.check <- c(arg.check, TRUE)**

**}**

**}**

**tempo <- fun\_check(data = y.top.extra.margin, prop = TRUE, length = 1, fun.name = function.name) ; eval(ee)**

**tempo <- fun\_check(data = y.bottom.extra.margin, prop = TRUE, length = 1, fun.name = function.name) ; eval(ee)**

**tempo <- fun\_check(data = xy.include.zero, class = "vector", mode = "logical", length = 1, fun.name = function.name) ; eval(ee)**

*# inactivated because xlim and ylim already log transformed*

*# if(tempo$problem == FALSE & ylog == TRUE & xy.include.zero == TRUE){*

*# warn.count <- warn.count + 1 ; tempo.warn <- paste0("(", warn.count,") FROM FUNCTION ", function.name, ": BOTH ylog AND xy.include.zero ARGUMENTS SET TO TRUE -> xy.include.zero ARGUMENT RESET TO FALSE")*

*# warn <- paste0(ifelse(is.null(warn), tempo.warn, paste0(warn, "\n\n", tempo.warn)))*

*# }*

**tempo <- fun\_check(data = text.size, class = "vector", mode = "numeric", length = 1, neg.values = FALSE, fun.name = function.name) ; eval(ee)**

**tempo <- fun\_check(data = title, class = "vector", mode = "character", length = 1, fun.name = function.name) ; eval(ee)**

**tempo <- fun\_check(data = title.text.size, class = "vector", mode = "numeric", length = 1, neg.values = FALSE, fun.name = function.name) ; eval(ee)**

**tempo <- fun\_check(data = show.legend, class = "logical", length = 1, fun.name = function.name) ; eval(ee)**

**tempo <- fun\_check(data = classic, class = "logical", length = 1, fun.name = function.name) ; eval(ee)**

**tempo <- fun\_check(data = grid, class = "logical", length = 1, fun.name = function.name) ; eval(ee)**

**tempo <- fun\_check(data = raster, class = "logical", length = 1, fun.name = function.name) ; eval(ee)**

**if( ! is.null(vectorial.limit)){**

**tempo <- fun\_check(data = vectorial.limit, class = "vector", typeof = "integer", neg.values = FALSE, double.as.integer.allowed = TRUE, fun.name = function.name) ; eval(ee)**

**}**

**tempo <- fun\_check(data = return, class = "logical", length = 1, fun.name = function.name) ; eval(ee)**

**tempo <- fun\_check(data = plot, class = "logical", length = 1, fun.name = function.name) ; eval(ee)**

**if( ! is.null(add)){**

**tempo <- fun\_check(data = add, class = "vector", mode = "character", length = 1, fun.name = function.name) ; eval(ee)**

**if(tempo$problem == FALSE & ! grepl(pattern = "^\\+", add)){** *# check that the add string start by +*

**tempo.cat <- paste0("ERROR IN ", function.name, ": add ARGUMENT MUST START WITH \"+\": ", paste(unique(add), collapse = " "))**

**text.check <- c(text.check, tempo.cat)**

**arg.check <- c(arg.check, TRUE)**

**}else if(tempo$problem == FALSE & ! grepl(pattern = "ggplot2::", add)){** *#*

**tempo.cat <- paste0("ERROR IN ", function.name, ": add ARGUMENT MUST CONTAIN \"ggplot2::\" IN FRONT OF EACH GGPLOT2 FUNCTION: ", paste(unique(add), collapse = " "))**

**text.check <- c(text.check, tempo.cat)**

**arg.check <- c(arg.check, TRUE)**

**}else if(tempo$problem == FALSE & ! grepl(pattern = ")$", add)){** *# check that the add string finished by )*

**tempo.cat <- paste0("ERROR IN ", function.name, ": add ARGUMENT MUST FINISH BY \")\": ", paste(unique(add), collapse = " "))**

**text.check <- c(text.check, tempo.cat)**

**arg.check <- c(arg.check, TRUE)**

**}**

**}**

**tempo <- fun\_check(data = warn.print, class = "logical", length = 1, fun.name = function.name) ; eval(ee)**

**if( ! is.null(lib.path)){**

**tempo <- fun\_check(data = lib.path, class = "vector", mode = "character", fun.name = function.name) ; eval(ee)**

**if(tempo$problem == FALSE & ! all(dir.exists(lib.path))){**

**tempo.cat <- paste0("ERROR IN ", function.name, ": \nDIRECTORY PATH INDICATED IN THE lib.path PARAMETER DOES NOT EXISTS: ", lib.path)**

**text.check <- c(text.check, tempo.cat)**

**arg.check <- c(arg.check, TRUE)**

**}**

**}**

**if(any(arg.check) == TRUE){**

**stop(paste0("\n\n================\n\n", paste(text.check[arg.check], collapse = "\n"), "\n\n================\n\n"), call. = FALSE)** *#*

**}**

*# source("C:/Users/Gael/Documents/Git\_versions\_to\_use/debugging\_tools\_for\_r\_dev-v1.2/r\_debugging\_tools-v1.2.R") ; eval(parse(text = str\_basic\_arg\_check\_dev)) ; eval(parse(text = str\_arg\_check\_with\_fun\_check\_dev)) # activate this line and use the function (with no arguments left as NULL) to check arguments status and if they have been checked using fun\_check()*

*# end argument checking*

*# package checking*

**fun\_pack(req.package = c("ggplot2"), lib.path = lib.path)**

*# packages Cairo and grid tested by fun\_gg\_point\_rast()*

*# end package checking*

*# main code*

*# axes management*

**if(is.null(xlim)){**

**if(any(unlist(mapply(FUN = "[[", data1, x, SIMPLIFY = FALSE)) %in% c(Inf, -Inf))){**

**warn.count <- warn.count + 1**

**tempo.warn <- paste0("(", warn.count,") FROM FUNCTION ", function.name, ": THE x COLUMN IN data1 CONTAINS -Inf OR Inf VALUES THAT WILL NOT BE CONSIDERED IN THE PLOT RANGE")**

**warn <- paste0(ifelse(is.null(warn), tempo.warn, paste0(warn, "\n\n", tempo.warn)))**

**}**

**xlim <- suppressWarnings(range(unlist(mapply(FUN = "[[", data1, x, SIMPLIFY = FALSE)), na.rm = TRUE, finite = TRUE))** *# finite = TRUE removes all the -Inf and Inf except if only this. In that case, whatever the -Inf and/or Inf present, output -Inf;Inf range. Idem with NA only. xlim added here. If NULL, ok if x argument has values*

**if(suppressWarnings(all(xlim %in% c(Inf, -Inf)))){**

**if(all(unlist(geom) == "geom\_hline")){**

**tempo.cat <- paste0("\n\n================\n\nERROR IN ", function.name, " NOT POSSIBLE TO ONLY DRAW geom\_hline KIND OF LINES IF xlim ARGUMENT IS SET TO NULL, SINCE NO X-AXIS DEFINED (", ifelse(length(x) == 1, "x", paste0("x NUMBER ", i1)), " ARGUMENT MUST BE NULL FOR THESE KIND OF LINES)\n\n================\n\n")**

**stop(tempo.cat, call. = FALSE)**

**}else{**

**tempo.cat <- paste0("\n\n================\n\nERROR IN ", function.name, " xlim ARGUMENT MADE OF NA, -Inf OR Inf ONLY: ", paste(xlim, collapse = " "), "\n\n================\n\n")**

**stop(tempo.cat, call. = FALSE)**

**}**

**}**

**}**

**xlim.order <- order(xlim)** *# to deal with inverse axis*

**# print(xlim.order)**

**xlim <- sort(xlim)**

**xlim[1] <- xlim[1] - abs(xlim[2] - xlim[1]) \* ifelse(diff(xlim.order) > 0, x.right.extra.margin, x.left.extra.margin)** *# diff(xlim.order) > 0 means not inversed axis*

**xlim[2] <- xlim[2] + abs(xlim[2] - xlim[1]) \* ifelse(diff(xlim.order) > 0, x.left.extra.margin, x.right.extra.margin)** *# diff(xlim.order) > 0 means not inversed axis*

**if(xy.include.zero == TRUE){** *# no need to check xlog != "no" because done before*

**xlim <- range(c(xlim, 0), na.rm = TRUE, finite = TRUE)** *# finite = TRUE removes all the -Inf and Inf except if only this. In that case, whatever the -Inf and/or Inf present, output -Inf;Inf range. Idem with NA only*

**}**

**xlim <- xlim[xlim.order]**

**if(any(is.na(xlim))){**

**tempo.cat <- paste0("\n\n============\n\nERROR IN ", function.name, ": CODE INCONSISTENCY 3\n\n============\n\n")**

**stop(tempo.cat, call. = FALSE)**

**}**

**if(is.null(ylim)){**

**if(any(unlist(mapply(FUN = "[[", data1, y, SIMPLIFY = FALSE)) %in% c(Inf, -Inf))){**

**warn.count <- warn.count + 1**

**tempo.warn <- paste0("(", warn.count,") FROM FUNCTION ", function.name, ": THE y COLUMN IN data1 CONTAINS -Inf OR Inf VALUES THAT WILL NOT BE CONSIDERED IN THE PLOT RANGE")**

**warn <- paste0(ifelse(is.null(warn), tempo.warn, paste0(warn, "\n\n", tempo.warn)))**

**}**

**ylim <- suppressWarnings(range(unlist(mapply(FUN = "[[", data1, y, SIMPLIFY = FALSE)), na.rm = TRUE, finite = TRUE))** *# finite = TRUE removes all the -Inf and Inf except if only this. In that case, whatever the -Inf and/or Inf present, output -Inf;Inf range. Idem with NA only. ylim added here. If NULL, ok if y argument has values*

**if(suppressWarnings(all(ylim %in% c(Inf, -Inf)))){** *# happen when y is only NULL*

**if(all(unlist(geom) == "geom\_vline")){**

**tempo.cat <- paste0("\n\n================\n\nERROR IN ", function.name, " NOT POSSIBLE TO ONLY DRAW geom\_vline KIND OF LINES IF ylim ARGUMENT IS SET TO NULL, SINCE NO Y-AXIS DEFINED (", ifelse(length(y) == 1, "y", paste0("y NUMBER ", i1)), " ARGUMENT MUST BE NULL FOR THESE KIND OF LINES)\n\n================\n\n")**

**stop(tempo.cat, call. = FALSE)**

**}else{**

**tempo.cat <- paste0("\n\n================\n\nERROR IN ", function.name, " ylim ARGUMENT MADE OF NA, -Inf OR Inf ONLY: ", paste(ylim, collapse = " "), "\n\n================\n\n")**

**stop(tempo.cat, call. = FALSE)**

**}**

**}**

**}**

**ylim.order <- order(ylim)** *# to deal with inverse axis*

**ylim <- sort(ylim)**

**ylim[1] <- ylim[1] - abs(ylim[2] - ylim[1]) \* ifelse(diff(ylim.order) > 0, y.bottom.extra.margin, y.top.extra.margin)** *# diff(ylim.order) > 0 means not inversed axis*

**ylim[2] <- ylim[2] + abs(ylim[2] - ylim[1]) \* ifelse(diff(ylim.order) > 0, y.top.extra.margin, y.bottom.extra.margin)** *# diff(ylim.order) > 0 means not inversed axis*

**if(xy.include.zero == TRUE){** *# no need to check ylog != "no" because done before*

**ylim <- range(c(ylim, 0), na.rm = TRUE, finite = TRUE)** *# finite = TRUE removes all the -Inf and Inf except if only this. In that case, whatever the -Inf and/or Inf present, output -Inf;Inf range. Idem with NA only*

**}**

**ylim <- ylim[ylim.order]**

**if(any(is.na(ylim))){**

**tempo.cat <- paste0("\n\n============\n\nERROR IN ", function.name, ": CODE INCONSISTENCY 4\n\n============\n\n")**

**stop(tempo.cat, call. = FALSE)**

**}**

*# end axes management*

*# create a fake categ if NULL to deal with legend display*

**if(is.null(categ)){**

**categ <- vector("list", length(data1))**

**categ[] <- "fake\_categ"**

**for(i2 in 1:length(data1)){**

**data1[[i2]] <- cbind(data1[[i2]], fake\_categ = "")**

**if(geom[[i2]] == "geom\_hline" | geom[[i2]] == "geom\_vline"){**

**data1[[i2]][, "fake\_categ"] <- paste0("Line\_", 1:nrow(data1[[i2]]))**

**}**

**}**

**warn.count <- warn.count + 1**

**tempo.warn <- paste0("(", warn.count,") FROM FUNCTION ", function.name, ": NULL categ ARGUMENT -> FAKE COLUMN ADDED TO EACH DATA FRAME IN data1, NAMED \"fake\_categ\" AND FILLED WITH \"\"")**

**warn <- paste0(ifelse(is.null(warn), tempo.warn, paste0(warn, "\n\n", tempo.warn)))**

**}**

*# end create a fake categ if NULL to deal with legend display*

*# vector of color with length as in data1*

**if(is.null(color)){**

**color <- vector("list", length(data1))**

**length.categ.list <- lapply(lapply(mapply(FUN = "[[", data1, categ, SIMPLIFY = FALSE), FUN = unique), FUN = function(x){length(x[ ! is.na(x)])})**

**length.categ.list[sapply(categ, FUN = "==", "fake\_categ")] <- 1** *# when is.null(color), a single color for all the dots or lines of data[[i1]] that contain "fake\_categ" category*

**total.categ.length <- sum(unlist(length.categ.list), na.rm = TRUE)**

**tempo.color <- fun\_gg\_palette(total.categ.length)**

**tempo.count <- 0**

**for(i3 in 1:length(data1)){**

**color[[i3]] <- tempo.color[(1:length.categ.list[[i3]]) + tempo.count]**

**tempo.count <- tempo.count + length.categ.list[[i3]]**

**warn.count <- warn.count + 1**

**tempo.warn <- paste0("(", warn.count,") FROM FUNCTION ", function.name, ": NULL color ARGUMENT -> COLORS RESPECTIVELY ATTRIBUTED TO EACH CLASS OF ", ifelse(length(categ) == 1, "categ", paste0("categ NUMBER ", i3)), " IN ", ifelse(length(data1) == 1, "data1", paste0("data1 NUMBER ", i3)), ":\n", paste(unlist(color), collapse = " "), "\n", paste(names(data1), collapse = " "))**

**warn <- paste0(ifelse(is.null(warn), tempo.warn, paste0(warn, "\n\n", tempo.warn)))**

**}**

**}**

*# end vector of color with length as in data1*

*# last check*

**for(i1 in 1:length(data1)){**

**if(categ[[i1]] != "fake\_categ" & length(color[[i1]]) != length(unique(data1[[i1]][, categ[[i1]]]))){**

**tempo.cat <- paste0("\n\n================\n\nERROR IN ", function.name, " LAST CHECK: ", ifelse(length(color) == 1, "color", paste0("color NUMBER ", i1)), " ARGUMENT MUST HAVE THE LENGTH OF LEVELS OF ", ifelse(length(categ) == 1, "categ", paste0("categ NUMBER ", i1)), " IN ", ifelse(length(data1) == 1, "data1", paste0("data1 NUMBER ", i1)), "\nHERE IT IS COLOR LENGTH ", length(color[[i1]]), " VERSUS CATEG LEVELS LENGTH ", length(unique(data1[[i1]][, categ[[i1]]])), "\n\n================\n\n")**

**stop(tempo.cat, call. = FALSE)**

**}else if(categ[[i1]] == "fake\_categ" & length(color[[i1]]) != 1){**

**tempo.cat <- paste0("\n\n================\n\nERROR IN ", function.name, " LAST CHECK: ", ifelse(length(color) == 1, "color", paste0("color NUMBER ", i1)), " ARGUMENT MUST HAVE LENGTH 1 WHEN ", ifelse(length(categ) == 1, "categ", paste0("categ NUMBER ", i1)), " IS NULL\nHERE IT IS COLOR LENGTH ", length(color[[i1]]), "\n\n================\n\n")**

**stop(tempo.cat, call. = FALSE)**

**}**

**}**

*# end last check*

*# conversion of geom\_hline and geom\_vline*

**for(i1 in 1:length(data1)){**

**if(geom[[i1]] == "geom\_hline" | geom[[i1]] == "geom\_vline"){**

**final.data.frame <- data.frame()**

**for(i3 in 1:nrow(data1[[i1]])){**

**tempo.data.frame <- rbind(data1[[i1]][i3, ], data1[[i1]][i3, ])**

**if(geom[[i1]] == "geom\_hline"){**

**tempo.data.frame[, x[[i1]]] <- xlim**

**}else if(geom[[i1]] == "geom\_vline"){**

**tempo.data.frame[, y[[i1]]] <- ylim**

**}else{**

**tempo.cat <- paste0("\n\n============\n\nERROR IN ", function.name, ": CODE INCONSISTENCY 5\n\n============\n\n")**

**stop(tempo.cat, call. = FALSE)**

**}**

**# if(is.null(categ[[i1]])){**

**# data1[, "fake\_categ"] <- paste0("Line\_", i3)**

**# } #I put that up**

**final.data.frame <- rbind(final.data.frame, tempo.data.frame)**

**}**

**data1[[i1]] <- final.data.frame**

**geom[[i1]] <- "geom\_line"**

**if(length(color[[i1]]) == 1){**

**color[[i1]] <- rep(color[[i1]], length(unique(data1[[i1]][ , categ[[i1]]])))**

**}else if(length(color[[i1]]) != length(unique(data1[[i1]][ , categ[[i1]]]))){**

**tempo.cat <- paste0("\n\n================\n\nERROR IN ", function.name, " geom\_hline AND geom\_vline CONVERSION TO FIT THE XLIM AND YLIM LIMITS OF THE DATA: ", ifelse(length(color) == 1, "color", paste0("color NUMBER ", i1)), " ARGUMENT MUST HAVE THE LENGTH OF LEVELS OF ", ifelse(length(categ) == 1, "categ", paste0("categ NUMBER ", i1)), " IN ", ifelse(length(data1) == 1, "data1", paste0("data1 NUMBER ", i1)), "\nHERE IT IS COLOR LENGTH ", length(color[[i1]]), " VERSUS CATEG LEVELS LENGTH ", length(unique(data1[[i1]][, categ[[i1]]])), "\n\n================\n\n")**

**stop(tempo.cat, call. = FALSE)**

**}**

**}**

**}**

*# end conversion of geom\_hline and geom\_vline*

*# kind of geom\_point (vectorial or raster)*

**scatter.kind <- vector("list", length = length(data1))** *# list of same length as data1, that will be used to use either ggplot2::geom\_point() (vectorial dot layer) or fun\_gg\_point\_rast() (raster dot layer)*

**fix.ratio <- FALSE**

**if(is.null(vectorial.limit)){**

**if(raster == TRUE){**

**scatter.kind[] <- "fun\_gg\_point\_rast"** *# not important to fill everything: will be only used when geom == "geom\_point"*

**fix.ratio <- TRUE**

**warn.count <- warn.count + 1**

**tempo.warn <- paste0("(", warn.count,") FROM FUNCTION ", function.name, ": RASTER PLOT GENERATED -> ASPECT RATIO OF THE PLOT REGION SET TO 1/1 TO AVOID A BUG OF ELLIPSOID DOT DRAWING")**

**warn <- paste0(ifelse(is.null(warn), tempo.warn, paste0(warn, "\n\n", tempo.warn)))**

**}else{**

**scatter.kind[] <- "ggplot2::geom\_point"**

**}**

**}else{**

**for(i2 in 1:length(data1)){**

**if(geom[[i2]] == "geom\_point"){**

**if(nrow(data1[[i2]]) <= vectorial.limit){**

**scatter.kind[[i2]] <- "ggplot2::geom\_point"**

**}else{**

**scatter.kind[[i2]] <- "fun\_gg\_point\_rast"**

**fix.ratio <- TRUE**

**warn.count <- warn.count + 1**

**tempo.warn <- paste0("(", warn.count,") FROM FUNCTION ", function.name, ": ", ifelse(length(data1) == 1, "data1", paste0("data1 NUMBER ", i2)), " LAYER AS RASTER (NOT VECTORIAL)")**

**warn <- paste0(ifelse(is.null(warn), tempo.warn, paste0(warn, "\n\n", tempo.warn)))**

**}**

**}**

**}**

**if(any(unlist(scatter.kind) == "fun\_gg\_point\_rast")){**

**warn.count <- warn.count + 1**

**tempo.warn <- paste0("(", warn.count,") FROM FUNCTION ", function.name, ": RASTER PLOT GENERATED -> ASPECT RATIO OF THE PLOT REGION SET TO 1/1 TO AVOID A BUG OF ELLIPSOID DOT DRAWING")**

**warn <- paste0(ifelse(is.null(warn), tempo.warn, paste0(warn, "\n\n", tempo.warn)))**

**}**

**}**

*# end kind of geom\_point (vectorial or raster)*

**tempo.gg.name <- "gg.indiv.plot."**

**tempo.gg.count <- 0**

*# no need loop part*

**assign(paste0(tempo.gg.name, tempo.gg.count <- tempo.gg.count + 1), ggplot2::ggplot())**

**assign(paste0(tempo.gg.name, tempo.gg.count <- tempo.gg.count + 1), ggplot2::xlab(if(is.null(xlab)){x[[1]]}else{xlab}))**

**assign(paste0(tempo.gg.name, tempo.gg.count <- tempo.gg.count + 1), ggplot2::ylab(if(is.null(ylab)){y[[1]]}else{ylab}))**

**assign(paste0(tempo.gg.name, tempo.gg.count <- tempo.gg.count + 1), ggplot2::ggtitle(title))**

**add.check <- TRUE**

**if( ! is.null(add)){** *# if add is NULL, then = 0*

**if(grepl(pattern = "ggplot2::theme", add) == TRUE){**

**warn.count <- warn.count + 1**

**tempo.warn <- paste0("(", warn.count,") FROM FUNCTION ", function.name, ": \"ggplot2::theme\" STRING DETECTED IN THE add ARGUMENT -> INTERNAL GGPLOT2 THEME FUNCTIONS theme() AND theme\_classic() HAVE BEEN INACTIVATED, TO BE USED BY THE USER.**

**\nIT IS RECOMMENDED TO USE \"+ theme(aspect.ratio = 1)\" IF RASTER MODE IS ACTIVATED")**

**warn <- paste0(ifelse(is.null(warn), tempo.warn, paste0(warn, "\n\n", tempo.warn)))**

**add.check <- FALSE**

**}**

**}**

**if(add.check == TRUE & classic == TRUE){**

*# BEWARE: not possible to add several times theme(). NO message but the last one overwrites the others*

**assign(paste0(tempo.gg.name, tempo.gg.count <- tempo.gg.count + 1), ggplot2::theme\_classic(base\_size = text.size))**

**if(grid == TRUE){**

**assign(paste0(tempo.gg.name, tempo.gg.count <- tempo.gg.count + 1), m.gg <- ggplot2::theme(**

**text = ggplot2::element\_text(size = text.size),**

**plot.title = ggplot2::element\_text(size = title.text.size),** *# stronger than text*

**line = ggplot2::element\_line(size = 0.5),**

**axis.line.y.left = ggplot2::element\_line(colour = "black"),** *# draw lines for the y axis*

**axis.line.x.bottom = ggplot2::element\_line(colour = "black"),** *# draw lines for the x axis*

**panel.grid.major.x = ggplot2::element\_line(colour = "grey75"),**

**panel.grid.major.y = ggplot2::element\_line(colour = "grey75"),**

**aspect.ratio = if(fix.ratio == TRUE){1}else{NULL}**

**))**

**}else{**

**assign(paste0(tempo.gg.name, tempo.gg.count <- tempo.gg.count + 1), m.gg <- ggplot2::theme(**

**text = ggplot2::element\_text(size = text.size),**

**plot.title = ggplot2::element\_text(size = title.text.size),** *# stronger than text*

**line = ggplot2::element\_line(size = 0.5),**

**axis.line.y.left = ggplot2::element\_line(colour = "black"),**

**axis.line.x.bottom = ggplot2::element\_line(colour = "black"),**

**aspect.ratio = if(fix.ratio == TRUE){1}else{NULL}**

**))**

**}**

**}else if(add.check == TRUE & classic == FALSE){**

**assign(paste0(tempo.gg.name, tempo.gg.count <- tempo.gg.count + 1), m.gg <- ggplot2::theme(**

**text = ggplot2::element\_text(size = text.size),**

**plot.title = ggplot2::element\_text(size = title.text.size),** *# stronger than text*

**line = ggplot2::element\_line(size = 0.5),**

**panel.background = ggplot2::element\_rect(fill = "grey95"),**

**axis.line.y.left = ggplot2::element\_line(colour = "black"),**

**axis.line.x.bottom = ggplot2::element\_line(colour = "black"),**

**panel.grid.major.x = ggplot2::element\_line(colour = "grey75"),**

**panel.grid.major.y = ggplot2::element\_line(colour = "grey75"),**

**panel.grid.minor.x = ggplot2::element\_blank(),**

**panel.grid.minor.y = ggplot2::element\_blank(),**

**strip.background = ggplot2::element\_rect(fill = "white", colour = "black"),**

**aspect.ratio = if(fix.ratio == TRUE){1}else{NULL}**

*# do not work -> legend.position = "none" # to remove the legend completely: https://www.datanovia.com/en/blog/how-to-remove-legend-from-a-ggplot/*

**))**

**}**

*# end no need loop part*

*# loop part*

**point.count <- 0**

**line.count <- 0**

**lg.order <- vector(mode = "list", length = 6)** *# order of the legend*

**lg.order <- lapply(lg.order, as.numeric)** *# order of the legend*

**lg.color <- vector(mode = "list", length = 6)** *# color of the legend*

**lg.alpha <- vector(mode = "list", length = 6)** *# order of the legend*

**lg.alpha <- lapply(lg.alpha, as.numeric)** *# alpha of the legend*

**for(i1 in 1:length(data1)){**

**if(geom[[i1]] == "geom\_point"){**

**point.count <- point.count + 1**

**if(point.count == 1){**

**fin.lg.disp[[1]] <- legend.disp[[point.count + line.count]]**

**lg.order[[1]] <- point.count + line.count**

**lg.color[[1]] <- color[[i1]]**

**lg.alpha[[1]] <- alpha[[i1]]**

**class.categ <- levels(factor(data1[[i1]][, categ[[i1]]]))**

**for(i5 in 1:length(color[[i1]])){** *# or length(class.categ). It is the same because already checked that lengths are the same*

**tempo.data.frame <- data1[[i1]][data1[[i1]][, categ[[i1]]] == class.categ[i5], ]**

**assign(paste0(tempo.gg.name, tempo.gg.count <- tempo.gg.count + 1), eval(parse(text = scatter.kind[[i1]]))(data = tempo.data.frame, mapping = ggplot2::aes\_string(x = x[[i1]], y = y[[i1]], fill = categ[[i1]]), size = dot.size, color = color[[i1]][i5], alpha = alpha[[i1]]))** *# beware: a single color allowed for color argument outside aesthetic, hence the loop # show.legend option do not remove the legend, only the aesthetic of the legend (dot, line, etc.)*

**}**

**assign(paste0(tempo.gg.name, tempo.gg.count <- tempo.gg.count + 1), ggplot2::scale\_fill\_manual(name = if(is.null(legend.name)){NULL}else{legend.name[[i1]]}, values = as.character(color[[i1]]), guide = ggplot2::guide\_legend(override.aes = list(colour = color[[i1]], linetype = 0))))** *# values are the values of fill. order determines the order in the legend*

**}**

**if(point.count == 2){**

**fin.lg.disp[[2]] <- legend.disp[[point.count + line.count]]**

**lg.order[[2]] <- point.count + line.count**

**lg.color[[2]] <- color[[i1]]**

**lg.alpha[[2]] <- alpha[[i1]]**

**class.categ <- levels(factor(data1[[i1]][, categ[[i1]]]))**

**for(i5 in 1:length(color[[i1]])){** *# or length(class.categ). It is the same because already checked that lengths are the same*

**tempo.data.frame <- data1[[i1]][data1[[i1]][, categ[[i1]]] == class.categ[i5], ]**

**assign(paste0(tempo.gg.name, tempo.gg.count <- tempo.gg.count + 1), eval(parse(text = scatter.kind[[i1]]))(data = tempo.data.frame, mapping = ggplot2::aes\_string(x = x[[i1]], y = y[[i1]], shape = categ[[i1]]), size = dot.size, color = color[[i1]][i5], alpha = alpha[[i1]]))** *# beware: a single color allowed for color argument outside aesthetic, hence the loop # show.legend option do not remove the legend, only the aesthetic of the legend (dot, line, etc.)*

**}**

**assign(paste0(tempo.gg.name, tempo.gg.count <- tempo.gg.count + 1), ggplot2::scale\_shape\_manual(name = if(is.null(legend.name)){NULL}else{legend.name[[i1]]}, values = rep(19, length(color[[i1]])), guide = ggplot2::guide\_legend(override.aes = list(colour = color[[i1]], linetype = 0))))** *# values are the values of shape*

**}**

**if(point.count == 3){**

**fin.lg.disp[[3]] <- legend.disp[[point.count + line.count]]**

**lg.order[[3]] <- point.count + line.count**

**lg.color[[3]] <- color[[i1]]**

**lg.alpha[[3]] <- alpha[[i1]]**

**class.categ <- levels(factor(data1[[i1]][, categ[[i1]]]))**

**for(i5 in 1:length(color[[i1]])){** *# or length(class.categ). It is the same because already checked that lengths are the same*

**tempo.data.frame <- data1[[i1]][data1[[i1]][, categ[[i1]]] == class.categ[i5], ]**

**assign(paste0(tempo.gg.name, tempo.gg.count <- tempo.gg.count + 1), eval(parse(text = scatter.kind[[i1]]))(data = tempo.data.frame, mapping = ggplot2::aes\_string(x = x[[i1]], y = y[[i1]], stroke = categ[[i1]]), size = dot.size, color = color[[i1]][i5], alpha = alpha[[i1]]))** *# beware: a single color allowed for color argument outside aesthetic, hence the loop # show.legend option do not remove the legend, only the aesthetic of the legend (dot, line, etc.)*

**}**

**assign(paste0(tempo.gg.name, tempo.gg.count <- tempo.gg.count + 1), ggplot2::scale\_discrete\_manual(aesthetics = "stroke", name = if(is.null(legend.name)){NULL}else{legend.name[[i1]]}, values = rep(0.5, length(color[[i1]])), guide = ggplot2::guide\_legend(override.aes = list(colour = color[[i1]], linetype = 0))))** *# values are the values of stroke*

**}**

**}else{**

**line.count <- line.count + 1**

**if(line.count == 1){**

**fin.lg.disp[[4]] <- legend.disp[[point.count + line.count]]**

**lg.order[[4]] <- point.count + line.count**

**lg.color[[4]] <- color[[i1]]**

**lg.alpha[[4]] <- 1** *# to avoid a bug on windows: if alpha argument is different from 1 for lines (transparency), then lines are not correctly displayed in the legend when using the R GUI (bug https://github.com/tidyverse/ggplot2/issues/2452). No bug when using a pdf*

**class.categ <- levels(factor(data1[[i1]][, categ[[i1]]]))**

**for(i5 in 1:length(color[[i1]])){** *# or length(class.categ). It is the same because already checked that lengths are the same*

**tempo.data.frame <- data1[[i1]][data1[[i1]][, categ[[i1]]] == class.categ[i5], ]**

**assign(paste0(tempo.gg.name, tempo.gg.count <- tempo.gg.count + 1), eval(parse(text = paste("ggplot2::", geom[[i1]], sep ="")))(data = tempo.data.frame, mapping = ggplot2::aes\_string(x = x[[i1]], y = y[[i1]], linetype = categ[[i1]]), color = color[[i1]][i5], size = line.size, lineend = "round", alpha = alpha[[i1]]))** *# beware: a single color allowed for color argument outside aesthetic, hence the loop # show.legend option do not remove the legend, only the aesthetic of the legend (dot, line, etc.)*

**}**

**assign(paste0(tempo.gg.name, tempo.gg.count <- tempo.gg.count + 1), ggplot2::scale\_discrete\_manual(aesthetics = "linetype", name = if(is.null(legend.name)){NULL}else{legend.name[[i1]]}, values = rep(1, length(color[[i1]])), guide = ggplot2::guide\_legend(override.aes = list(colour = color[[i1]], shape = NA))))** *# values are the values of linetype. 1 means solid. Regarding the alpha bug, I have tried different things without success: alpha in guide alone, in geom alone, in both, with different values*

**}**

**if(line.count == 2){**

**fin.lg.disp[[5]] <- legend.disp[[point.count + line.count]]**

**lg.order[[5]] <- point.count + line.count**

**lg.color[[5]] <- color[[i1]]**

**lg.alpha[[5]] <- 1** *# to avoid a bug on windows: if alpha argument is different from 1 for lines (transparency), then lines are not correctly displayed in the legend when using the R GUI (bug https://github.com/tidyverse/ggplot2/issues/2452). No bug when using a pdf*

**class.categ <- levels(factor(data1[[i1]][, categ[[i1]]]))**

**for(i5 in 1:length(color[[i1]])){** *# or length(class.categ). It is the same because already checked that lengths are the same*

**tempo.data.frame <- data1[[i1]][data1[[i1]][, categ[[i1]]] == class.categ[i5], ]**

**assign(paste0(tempo.gg.name, tempo.gg.count <- tempo.gg.count + 1), eval(parse(text = paste("ggplot2::", geom[[i1]], sep ="")))(data = tempo.data.frame, mapping = ggplot2::aes\_string(x = x[[i1]], y = y[[i1]], alpha = categ[[i1]]), color = color[[i1]][i5], size = line.size, lineend = "round"))** *# beware: a single color allowed for color argument outside aesthetic, hence the loop # show.legend option do not remove the legend, only the aesthetic of the legend (dot, line, etc.)*

**}**

**assign(paste0(tempo.gg.name, tempo.gg.count <- tempo.gg.count + 1), ggplot2::scale\_discrete\_manual(aesthetics = "alpha", name = if(is.null(legend.name)){NULL}else{legend.name[[i1]]}, values = rep(alpha[[i1]], length(color[[i1]])), guide = ggplot2::guide\_legend(override.aes = list(colour = color[[i1]], shape = NA))))** *# values are the values of linetype. 1 means solid. Regarding the alpha bug, I have tried different things without success: alpha in guide alone, in geom alone, in both, with different values*

**}**

**if(line.count == 3){**

**fin.lg.disp[[6]] <- legend.disp[[point.count + line.count]]**

**lg.order[[6]] <- point.count + line.count**

**lg.color[[6]] <- color[[i1]]**

**lg.alpha[[6]] <- 1** *# to avoid a bug on windows: if alpha argument is different from 1 for lines (transparency), then lines are not correctly displayed in the legend when using the R GUI (bug https://github.com/tidyverse/ggplot2/issues/2452). No bug when using a pdf*

**class.categ <- levels(factor(data1[[i1]][, categ[[i1]]]))**

**for(i5 in 1:length(color[[i1]])){** *# or length(class.categ). It is the same because already checked that lengths are the same*

**tempo.data.frame <- data1[[i1]][data1[[i1]][, categ[[i1]]] == class.categ[i5], ]**

**assign(paste0(tempo.gg.name, tempo.gg.count <- tempo.gg.count + 1), eval(parse(text = paste("ggplot2::", geom[[i1]], sep ="")))(data = tempo.data.frame, mapping = ggplot2::aes\_string(x = x[[i1]], y = y[[i1]], size = categ[[i1]]), color = color[[i1]][i5], alpha = alpha[[i1]], lineend = "round"))** *# beware: a single color allowed for color argument outside aesthetic, hence the loop # show.legend option do not remove the legend, only the aesthetic of the legend (dot, line, etc.)*

**}**

**assign(paste0(tempo.gg.name, tempo.gg.count <- tempo.gg.count + 1), ggplot2::scale\_discrete\_manual(aesthetics = "size", name = if(is.null(legend.name)){NULL}else{legend.name[[i1]]}, values = rep(line.size, length(color[[i1]])), guide = ggplot2::guide\_legend(override.aes = list(colour = color[[i1]], shape = NA))))** *# values are the values of linetype. 1 means solid. Regarding the alpha bug, I have tried different things without success: alpha in guide alone, in geom alone, in both, with different values*

**}**

**}**

**}**

*# end loop part*

*# legend display*

**assign(paste0(tempo.gg.name, tempo.gg.count <- tempo.gg.count + 1), ggplot2::guides(fill = if(fin.lg.disp[[1]] == TRUE){ggplot2::guide\_legend(order = lg.order[[1]], override.aes = list(alpha = lg.alpha[[1]], color = lg.color[[1]]))}else{FALSE}, shape = if(fin.lg.disp[[2]] == TRUE){ggplot2::guide\_legend(order = lg.order[[2]], override.aes = list(alpha = lg.alpha[[2]], color = lg.color[[2]]))}else{FALSE}, stroke = if(fin.lg.disp[[3]] == TRUE){ggplot2::guide\_legend(order = lg.order[[3]], override.aes = list(alpha = lg.alpha[[2]], color = lg.color[[3]]))}else{FALSE}, linetype = if(fin.lg.disp[[4]] == TRUE){ggplot2::guide\_legend(order = lg.order[[4]], override.aes = list(alpha = lg.alpha[[4]], color = lg.color[[4]]))}else{FALSE}, alpha = if(fin.lg.disp[[5]] == TRUE){ggplot2::guide\_legend(order = lg.order[[5]], override.aes = list(alpha = lg.alpha[[5]], color = lg.color[[5]]))}else{FALSE}, size = if(fin.lg.disp[[6]] == TRUE){ggplot2::guide\_legend(order = lg.order[[6]], override.aes = list(alpha = lg.alpha[[6]], color = lg.color[[6]]))}else{FALSE}))** *# clip = "off" to have secondary ticks outside plot region does not work*

*# end legend display*

*# scale management*

**assign(paste0(tempo.gg.name, tempo.gg.count <- tempo.gg.count + 1), ggplot2::coord\_cartesian(xlim = xlim, ylim = ylim))** *# clip = "off" to have secondary ticks outside plot region does not work*

*# x-axis ticks and inv*

**tempo.coord <- ggplot2::ggplot\_build(eval(parse(text = paste(paste0(tempo.gg.name, 1:tempo.gg.count), collapse = " + "))))$layout$panel\_params[[1]]**

**tempo.scale <- fun\_scale(lim = xlim, n = ifelse(is.null(x.tick.nb), length(tempo.coord$x.major\_source), x.tick.nb))**

**assign(paste0(tempo.gg.name, tempo.gg.count <- tempo.gg.count + 1), ggplot2::scale\_x\_continuous(**

**breaks = tempo.scale,**

**labels = if(xlog == "log10"){scales::trans\_format("identity", scales::math\_format(10^.x))}else if(xlog == "log2"){scales::trans\_format("identity", scales::math\_format(2^.x))}else if(xlog == "no"){ggplot2::waiver()}else{tempo.cat <- paste0("\n\n============\n\nERROR IN ", function.name, ": CODE INCONSISTENCY 6\n\n============\n\n") ; stop(tempo.cat, call. = FALSE)},**

**expand = c(0, 0),**

**limits = NA,**

**trans = ifelse(diff(xlim) < 0, "reverse", "identity")** *# equivalent to ggplot2::scale\_x\_reverse()*

**))**

*# end x-axis ticks and inv*

*# y-axis ticks and inv*

**tempo.coord <- ggplot2::ggplot\_build(eval(parse(text = paste(paste0(tempo.gg.name, 1:tempo.gg.count), collapse = " + "))))$layout$panel\_params[[1]]**

**tempo.scale <- fun\_scale(lim = ylim, n = ifelse(is.null(y.tick.nb), length(tempo.coord$y.major\_source), y.tick.nb))**

**assign(paste0(tempo.gg.name, tempo.gg.count <- tempo.gg.count + 1), ggplot2::scale\_y\_continuous(**

**breaks = tempo.scale,**

**labels = if(ylog == "log10"){scales::trans\_format("identity", scales::math\_format(10^.x))}else if(ylog == "log2"){scales::trans\_format("identity", scales::math\_format(2^.x))}else if(ylog == "no"){ggplot2::waiver()}else{tempo.cat <- paste0("\n\n============\n\nERROR IN ", function.name, ": CODE INCONSISTENCY 7\n\n============\n\n") ; stop(tempo.cat, call. = FALSE)},**

**expand = c(0, 0),**

**limits = NA,**

**trans = ifelse(diff(ylim) < 0, "reverse", "identity")** *# equivalent to ggplot2::scale\_y\_reverse()*

**))**

*# end y-axis ticks and inv*

*# x-axis secondary ticks (after ggplot2::coord\_cartesian() or ggplot2::coord\_flip())*

**tempo.coord <- ggplot2::ggplot\_build(eval(parse(text = paste(paste0(tempo.gg.name, 1:tempo.gg.count), collapse = " + "))))$layout$panel\_params[[1]]**

**xlim.order <- order(xlim)** *# to deal with inverse axis*

**ylim.order <- order(ylim)** *# to deal with inverse axis*

*# no secondary ticks for log2. Play with xlim*

**if(xlog == "log10"){**

**y.range <- tempo.coord$y.range**

**if(diff(ylim.order) < 0){y.range <- -(y.range)}**

**ini.scipen <- options()$scipen**

**options(scipen = -1000)** *# force scientific format*

**power10.exp <- as.integer(substring(text = 10^xlim, first = (regexpr(pattern = "\\+|\\-", text = 10^xlim))))** *# recover the power of 10. Example recover 08 from 1e+08*

**# print(xlim)**

**mantisse <- as.numeric(substr(x = 10^xlim, start = 1, stop = (regexpr(pattern = "\\+|\\-", text = 10^xlim) - 2)))** *# recover the mantisse. Example recover 1.22 from 1.22e+08*

**options(scipen = ini.scipen)** *# restore the initial scientific penalty*

**# print(power10.exp)**

**tempo.tick.pos <- as.vector(outer(log10(2:10), 10^((power10.exp[1] - ifelse(diff(xlim.order) > 0, 1, -1)):(power10.exp[2] + ifelse(diff(xlim.order) > 0, 1, -1)))))**

**tempo.tick.pos <- sort(tempo.tick.pos, decreasing = ifelse(diff(xlim.order) > 0, FALSE, TRUE))**

**tempo.tick.pos <- log10(tempo.tick.pos[tempo.tick.pos >= min(10^xlim) & tempo.tick.pos <= max(10^xlim)])**

**if(any(is.na(tempo.tick.pos) | ! is.finite(tempo.tick.pos))){**

**tempo.cat <- paste0("\n\n============\n\nERROR IN ", function.name, ": CODE INCONSISTENCY 8\n\n============\n\n")**

**stop(tempo.cat, call. = FALSE)**

**}**

**assign(paste0(tempo.gg.name, tempo.gg.count <- tempo.gg.count + 1), ggplot2::annotate(geom = "segment", x = tempo.tick.pos, xend = tempo.tick.pos, y = y.range[1], yend = y.range[1] + diff(y.range) / 80))**

**}else if(( ! is.null(x.inter.tick.nb)) & xlog == "no"){**

**if(x.inter.tick.nb > 0){**

**x.ticks.pos <- suppressWarnings(as.numeric(tempo.coord$x.labels))** *# too difficult to predict the behavior of tempo.coord$x.major\_source depending on xlim neg or not, inv or not. Inv is respected*

**if(any(is.na(x.ticks.pos))){**

**tempo.cat <- paste0("\n\n============\n\nERROR IN ", function.name, ": CODE INCONSISTENCY 9\n\n============\n\n")**

**stop(tempo.cat, call. = FALSE)**

**}**

**y.range <- tempo.coord$y.range**

**if(diff(ylim.order) < 0){y.range <- -(y.range)}**

**tick.dist <- mean(diff(x.ticks.pos), na.rm = TRUE)**

**minor.tick.dist <- tick.dist / (x.inter.tick.nb + 1)**

**minor.tick.pos <- seq(x.ticks.pos[1] - tick.dist, x.ticks.pos[length(x.ticks.pos)] + tick.dist, by = minor.tick.dist)**

**assign(paste0(tempo.gg.name, tempo.gg.count <- tempo.gg.count + 1), ggplot2::annotate(geom = "segment", x = minor.tick.pos, xend = minor.tick.pos, y = y.range[1], yend = y.range[1] + diff(y.range) / 80))**

**}**

**}**

*# end x-axis secondary ticks (after ggplot2::coord\_cartesian() or ggplot2::coord\_flip())*

*# y-axis secondary ticks (after ggplot2::coord\_cartesian() or ggplot2::coord\_flip())*

**tempo.coord <- ggplot2::ggplot\_build(eval(parse(text = paste(paste0(tempo.gg.name, 1:tempo.gg.count), collapse = " + "))))$layout$panel\_params[[1]]**

*# no secondary ticks for log2. Play with ylim*

**if(ylog == "log10"){**

**x.range <- tempo.coord$x.range**

**if(diff(xlim.order) < 0){x.range <- -(x.range)}**

**ini.scipen <- options()$scipen**

**options(scipen = -1000)** *# force scientific format*

**power10.exp <- as.integer(substring(text = 10^ylim, first = (regexpr(pattern = "\\+|\\-", text = 10^ylim))))** *# recover the power of 10. Example recover 08 from 1e+08*

**mantisse <- as.numeric(substr(x = 10^ylim, start = 1, stop = (regexpr(pattern = "\\+|\\-", text = 10^ylim) - 2)))** *# recover the mantisse. Example recover 1.22 from 1.22e+08*

**options(scipen = ini.scipen)** *# restore the initial scientific penalty*

**tempo.tick.pos <- as.vector(outer(log10(2:10), 10^((power10.exp[1] - ifelse(diff(ylim.order) > 0, 1, -1)):(power10.exp[2] + ifelse(diff(ylim.order) > 0, 1, -1)))))**

**tempo.tick.pos <- sort(tempo.tick.pos, decreasing = ifelse(diff(ylim.order) > 0, FALSE, TRUE))**

**tempo.tick.pos <- log10(tempo.tick.pos[tempo.tick.pos >= min(10^ylim) & tempo.tick.pos <= max(10^ylim)])**

**if(any(is.na(tempo.tick.pos) | ! is.finite(tempo.tick.pos))){**

**tempo.cat <- paste0("\n\n============\n\nERROR IN ", function.name, ": CODE INCONSISTENCY 10\n\n============\n\n")**

**stop(tempo.cat, call. = FALSE)**

**}**

**assign(paste0(tempo.gg.name, tempo.gg.count <- tempo.gg.count + 1), ggplot2::annotate(geom = "segment", y = tempo.tick.pos, yend = tempo.tick.pos, x = x.range[1], xend = x.range[1] + diff(x.range) / 80))**

**}else if(( ! is.null(y.inter.tick.nb)) & ylog == "no"){**

**if(y.inter.tick.nb > 0){**

**y.ticks.pos <- suppressWarnings(as.numeric(tempo.coord$y.labels))** *# too difficult to predict the behavior of tempo.coord$y.major\_source depending on ylim neg or not, inv or not. Inv is respected*

**if(any(is.na(y.ticks.pos))){**

**tempo.cat <- paste0("\n\n============\n\nERROR IN ", function.name, ": CODE INCONSISTENCY 11\n\n============\n\n")**

**stop(tempo.cat, call. = FALSE)**

**}**

**x.range <- tempo.coord$x.range**

**if(diff(xlim.order) < 0){x.range <- -(x.range)}**

**tick.dist <- mean(diff(y.ticks.pos), na.rm = TRUE)**

**minor.tick.dist <- tick.dist / (y.inter.tick.nb + 1)**

**minor.tick.pos <- seq(y.ticks.pos[1] - tick.dist, y.ticks.pos[length(y.ticks.pos)] + tick.dist, by = minor.tick.dist)**

**assign(paste0(tempo.gg.name, tempo.gg.count <- tempo.gg.count + 1), ggplot2::annotate(geom = "segment", y = minor.tick.pos, yend = minor.tick.pos, x = x.range[1], xend = x.range[1] + diff(x.range) / 80))**

**}**

**}**

*# end y-axis secondary ticks (after ggplot2::coord\_cartesian() or ggplot2::coord\_flip())*

*# end scale management*

**if(plot == TRUE){**

**suppressWarnings(print(eval(parse(text = paste(paste(paste0(tempo.gg.name, 1:tempo.gg.count), collapse = " + "),if(is.null(add)){NULL}else{add})))))**

**}else{**

**warn.count <- warn.count + 1**

**tempo.warn <- paste0("(", warn.count,") FROM FUNCTION ", function.name, ": PLOT NOT SHOWN AS REQUESTED")**

**warn <- paste0(ifelse(is.null(warn), tempo.warn, paste0(warn, "\n\n", tempo.warn)))**

**}**

**if(warn.print == TRUE & ! is.null(warn)){**

**warning(warn, call. = FALSE)**

**cat("\n\n")**

**}**

**if(return == TRUE){**

**output <- ggplot2::ggplot\_build(eval(parse(text = paste(paste0(tempo.gg.name, 1:tempo.gg.count), collapse = " + "))))**

**if(is.null(unlist(removed.row.nb))){**

**removed.row.nb <- NULL**

**removed.rows <- NULL**

**}else{**

**for(i3 in 1:length(data1)){**

**if( ! is.null(removed.row.nb[[i3]])){**

**removed.row.nb[[i3]] <- sort(removed.row.nb[[i3]])**

**removed.rows[[i3]] <- data1.ini[[i3]][removed.row.nb[[i3]], ]**

**}**

**}**

**}**

**output <- list(data = output$data, removed.row.nb = removed.row.nb, removed.rows = removed.rows, axes = output$layout$panel\_params[[1]], warn = paste0("\n", warn, "\n\n"))**

**return(output)**

**}**

**}**