**# DO NOT ERASE. COMPARE WITH BAR MEAN BEFORE AND RECOVER WHAT HAS BEEN MODIFIED**

**fun\_gg\_line <- function(data1, y, categ, categ.class.order = NULL, categ.legend.name = NULL, categ.color = NULL, line.size = 1, error.disp = NULL, error.whisker.width = 0.5, dot.color = "same", dot.tidy = FALSE, dot.bin.nb = 30, dot.jitter = 0.25, dot.size = 3, dot.border.size = 0.5, dot.alpha = 0.5, ylim = NULL, ylog = FALSE, y.tick.nb = NULL, y.include.zero = FALSE, y.top.extra.margin = 0.05, y.bottom.extra.margin = 0, stat.disp = NULL, stat.size = 4, stat.dist = 2, xlab = NULL, ylab = NULL, vertical = TRUE, title = "", text.size = 12, text.angle = 0, classic = FALSE, grid = FALSE, return = FALSE, lib.path = NULL){**

*# AIM*

*# ggplot2 vertical barplot representing mean values with the possibility to add error bars and to overlay dots*

*# 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)*

*# if ever bars disappear, see the end of https://github.com/tidyverse/ggplot2/issues/2887*

*# to have a single bar, create a factor column with a single class and specify the name of this column in categ argument as unique element (no categ2 in categ argument). For a single set of grouped bars, create a factor column with a single class and specify this column in categ argument as first element (categ1). See categ below*

*# with several single bars (categ argument with only one element), bar.width argument (i.e., width argument of ggplot2::geom\_bar()) defines each bar width. The bar.width argument also defines the space between bars by using (1 - bar.width). In addition, xmin and xmax of the fun\_gg\_bar() output report the bar boundaries (around x-axis unit 1, 2, 3, etc., for each bar)*

*# with several sets of grouped bars (categ argument with two elements), bar.width argument defines each set of grouped bar width. The bar.width argument also defines the space between set of grouped bars by using (1 - bar.width). In addition, xmin and xmax of the fun\_gg\_bar() output report the bar boundaries (around x-axis unit 1, 2, 3, etc., for each set of grouped bar)*

*# to manually change the 0 base bar into this code, see https://stackoverflow.com/questions/35324892/ggplot2-setting-geom-bar-baseline-to-1-instead-of-zero*

*# ARGUMENTS*

*# data1: a dataframe containing one column of values (see y argument below) and one or two columns of categories (see categ argument below). Duplicated column names not allowed*

*# y: character string of the data1 column name for y-axis (containing numeric values). Numeric values will be averaged by categ to generate the bars and will also be used to plot the dots*

*# categ: vector of character strings of the data1 column name for categories (column of characters or factor). Must either be one or two column names. If a single column name (further refered to as categ1), then one bar per class of categ1. If two column names (further refered to as categ1 and categ2), then one bar per class of categ2, which form a group of bars in each class of categ1. BEWARE, categ1 (and categ2 if it exists) must have a single value of y per class of categ1 (and categ2). To have a single bar, create a factor column with a single class and specify the name of this column in categ argument as unique element (no categ2 in categ argument). For a single set of grouped bars, create a factor column with a single class and specify this column in categ argument as first element (categ1)*

*# categ.class.order: list indicating the order of the classes of categ1 and categ2 represented on the barplot (the first compartment for categ1 and and the second for categ2). If categ.class.order = NULL, classes are represented according to the alphabetical order. Some compartment can be NULL and other not*

*# categ.legend.name: character string of the legend title for categ2. If categ.legend.name = NULL, then categ.legend.name <- categ1 if only categ1 is present and categ.legend.name <- categ2 if categ1 and categ2 are present. Write "" if no legend required*

*# categ.color: vector of character color string for bar filling. If categ.color = NULL, default colors of ggplot2, whatever categ1 and categ2. If categ.color is non null and only categ1 in categ argument, categ.color can be either: (1) a single color string (all the bars will have this color, whatever the classes of categ1), (2) a vector of string colors, one for each class of categ1 (each color will be associated according to categ.class.order of categ1), (3) 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 categ1 and a single class of categ1 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 categ.color. If categ.color is non null and categ1 and categ2 specified, all the rules described above will apply to categ2 instead of categ1 (colors will be determined for bars inside a group of bars)*

*# bar.width: numeric value (from 0 to 1) of the bar or set of grouped bar width (see WARNINGS above)*

*# error.disp: either "SD", "SD.TOP", "SEM" or "SEM.TOP". If NULL, no error bars added*

*# error.whisker.width: numeric value (from 0 to 1) of the whisker (error bar extremities) width, with 0 meaning no whiskers and 1 meaning a width equal to the corresponding bar width*

*# dot.color: vector of character string. Idem as categ.color but for dots, except that in the possibility (3), the rule "a single color per class of categ1 and a single class of categ1", cannot be respected (each dot can have a different color). If NULL, no dots plotted*

*# dot.tidy: logical. Nice dot spreading? If TRUE, use the geom\_dotplot() function for a nice representation. If FALSE, dots are randomly spread, using the dot.jitter argument (see below)*

*# dot.bin.nb: positive integer indicating the number of bins (i.e., nb of separations) of the ylim range. Each dot will then be put in one of the bin, with the size the width of the bin. Not considered if dot.tidy is FALSE*

*# dot.jitter: numeric value (from 0 to 1) of random dot horizontal dispersion, with 0 meaning no dispersion and 1 meaning a dispersion in the corresponding bar width interval. Not considered if dot.tidy is TRUE*

*# dot.size: numeric value of dot size. Not considered if dot.tidy is TRUE*

*# dot.border.size: numeric value of border dot size. Write zero for no dot border. If dot.tidy is TRUE, value 0 remove the border. Another one leave the border without size control (geom\_doplot() feature)*

*# dot.alpha: numeric value (from 0 to 1) of dot transparency (full transparent to full opaque, respectively)*

*# ylim: 2 numeric values for y-axis range. If NULL, range of y in data1*

*# ylog: logical. Log scale for the y-axis? BEWARE: do not tranform the data, but just display ticks in a log scale manner. BEWARE: if TRUE, ylim must not contain null or negative values. In addition, will be automatically set to FALSE if vertical argument is set to FALSE, to prevent a bug in ggplot2 (see https://github.com/tidyverse/ggplot2/issues/881)*

*# y.tick.nb: number of desired values on the y-axis*

*# y.include.zero: logical. Does ylim range include 0? BEWARE: if ylog = TRUE, will be automately set to FALSE with a warning message*

*# 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. BEWARE with ylog = TRUE, the range result must not overlap zero or negative values*

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

*# stat.disp: add the mean number above the corresponding bar. Either NULL (no number shown), "top" (at the top of the figure region) or "above" (above each bar)*

*# stat.size: numeric value of the stat size (in points). Increase the value to increase text size*

*# stat.dist: numeric value of the stat distance. Increase the value to increase the distance*

*# xlab: a character string for x-axis legend. If NULL, character string of categ1*

*# ylab: a character string y-axis legend. If NULL, character string of the y argument*

*# vertical: logical. Vertical bars? BEWARE: cannot have horizontal bars with a log axis, i.e., ylog = TRUE & vertical = FALSE (see ylog above)*

*# title: character string of the graph title*

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

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

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

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

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

*# lib.path: absolute path of the required packages, if not in the default folders*

**}**

######## fun\_gg\_heatmap() #### ggplot2 heatmap + overlaid mask if required

**#test plot.margin = margin(up.space.mds, right.space.mds, down.space.mds, left.space.mds, "inches") to set the dim of the region plot ?**

**# if matrix is full of zero (or same value I guess), heatmap is complicate. Test it and error message**

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

**fun\_gg\_heatmap <- function(data1, legend.name1 = "", low.color1 = "blue", mid.color1 = "white", high.color1 = "red", limit1 = NULL, midpoint1 = NULL, data2 = NULL, color2 = "black", alpha2 = 0.5, invert2 = FALSE, text.size = 12, title = "", title.text.size = 12, show.scale = TRUE, rotate = FALSE, return = FALSE, plot = TRUE, add = NULL, warn.print = FALSE, lib.path = NULL){**

*# AIM*

*# ggplot2 heatmap with the possibility to overlay a mask*

*# see also:*

*# draw : http://www.sthda.com/english/wiki/ggplot2-quick-correlation-matrix-heatmap-r-software-and-data-visualization*

*# same range scale : https://stackoverflow.com/questions/44655723/r-ggplot2-heatmap-fixed-scale-color-between-graphs*

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

*# ARGUMENTS*

*# data1: numeric matrix or data frame resulting from the conversion of the numeric matrix by reshape2::melt()*

*# legend.name1: character string of the data1 heatmap scale legend*

*# low.color1: character string of the color (i.e., "blue" or "#0000FF") of the lowest scale value*

*# mid.color1: same as low.color1 but for the middle scale value. If NULL, the middle color is the default color between low.color1 and high.color1. BEWARE: argument midpoint1 is not ignored, even if mid.color1 is NULL, meaning that the default mid color can still be controled*

*# high.color1: same as low.color1 but for the highest scale value*

*# limit1: 2 numeric values defining the lowest and higest color scale values. If NULL, take the range of data1 values*

*# midpoint1: single numeric value defining the value corresponding to the mid.color1 argument. A warning message is returned if midpoint1 does not correspond to the mean of limit1 values, because the color scale is not linear anymore. If NULL, takes the mean of limit1 values. Mean of data1, instead of mean of limit1, can be used here if required*

*# data2: binary mask matrix (made of 0 and 1) of same dimension as data1 or a data frame resulting from the conversion of the binary mask matrix by reshape2::melt(). Value 1 of data2 will correspond to color2 argument (value 0 will be NA color), and the opposite if invert2 argument is TRUE (inverted mask)*

*# color2: color of the 1 values of the binary mask matrix. The 0 values will be color NA*

*# alpha2: numeric value (from 0 to 1) of the mask transparency*

*# invert2: logical. Invert the mask (1 -> 0 and 0 -> 1)?*

*# text.size: numeric value of the size of the texts in scale*

*# title: character string of the graph title*

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

*# show.scale: logical. Show color scale?*

*# rotate: logical. Rotate the heatmap 90° clockwise?*

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

*# 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*

*# reshape2*

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

*# fun\_check()*

*# fun\_pack()*

*# fun\_round()*

*# RETURN*

*# a heatmap if plot argument is TRUE*

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

*# $data: a list of the graphic info*

*# $axes: a list of the axes info*

*# $scale: the scale info (lowest, mid and highest values)*

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

*# EXAMPLES*

*# fun\_gg\_heatmap(data1 = matrix(1:16, ncol = 4), title = "GRAPH 1")*

*# fun\_gg\_heatmap(data1 = matrix(1:16, ncol = 4), return = TRUE)*

*# fun\_gg\_heatmap(data1 = matrix(1:16, ncol = 4), legend.name1 = "VALUE", title = "GRAPH 1", text.size = 5, data2 = matrix(rep(c(1,0,0,0), 4), ncol = 4), invert2 = FALSE, return = TRUE)*

*# diagonal matrix*

*# fun\_gg\_heatmap(data1 = matrix(c(1,0,0,0,0,1,0,0,0,0,1,0,0,0,0,1), ncol = 4))*

*# fun\_gg\_heatmap(data1 = reshape2::melt(matrix(c(1,0,0,0,0,1,0,0,0,0,1,0,0,0,0,1), ncol = 4)))*

*# error message*

*# fun\_gg\_heatmap(data1 = matrix(1:16, ncol = 4), data2 = matrix(rep(c(1,0,0,0), 5), ncol = 5))*

*# fun\_gg\_heatmap(data1 = matrix(1:16, ncol = 4), data2 = reshape2::melt(matrix(rep(c(1,0,0,0), 4), ncol = 4)))*

*# fun\_gg\_heatmap(data1 = reshape2::melt(matrix(1:16, ncol = 4)), data2 = reshape2::melt(matrix(rep(c(1,0,0,0), 4), ncol = 4)))*

*#### 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*

*# data1 = matrix(1:16, ncol = 4) ; legend.name1 = "" ; low.color1 = "blue" ; mid.color1 = "white" ; high.color1 = "red" ; limit1 = NULL ; midpoint1 = NULL ; data2 = matrix(rep(c(1,0,0,0), 4), ncol = 4) ; color2 = "black" ; alpha2 = 0.5 ; invert2 = FALSE ; text.size = 12 ; title = "" ; title.text.size = 12 ; show.scale = TRUE ; rotate = FALSE ; 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*

**if(length(utils::find("fun\_check", mode = "function")) == 0){**

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

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

**}**

**if(length(utils::find("fun\_pack", mode = "function")) == 0){**

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

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

**}**

**if(length(utils::find("fun\_round", mode = "function")) == 0){**

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

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

**}**

*# end required function checking*

*# no reserved words required for this function*

*# argument checking*

**warn <- NULL**

**warn.count <- 0**

**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))**

**if(all(is.matrix(data1))){**

**tempo <- fun\_check(data = data1, class = "matrix", mode = "numeric", na.contain = TRUE, fun.name = function.name) ; eval(ee)**

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

**tempo <- fun\_check(data = data1, class = "data.frame", length = 3, fun.name = function.name) ; eval(ee)**

**if(tempo$problem == FALSE){**

*# structure of reshape2::melt() data frame*

**tempo <- fun\_check(data = data1[, 1], typeof = "integer", fun.name = function.name)**

**tempo <- fun\_check(data = data1[, 2], typeof = "integer", fun.name = function.name)**

**tempo <- fun\_check(data = data1[, 3], mode = "numeric", na.contain = TRUE, fun.name = function.name)**

**}**

**}else{**

**tempo.cat <- paste0("ERROR IN ", function.name, ": THE data1 ARGUMENT MUST BE A NUMERIC MATRIX OR A DATA FRAME OUTPUT OF THE reshape::melt() FUNCTION\n\n================\n\n")**

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

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

**}**

**tempo <- fun\_check(data = legend.name1, class = "character", length = 1, fun.name = function.name) ; eval(ee)**

**tempo <- fun\_check(data = low.color1, class = "character", length = 1, fun.name = function.name) ; eval(ee)**

**if(tempo$problem == FALSE & ! (all(low.color1 %in% colors() | grepl(pattern = "^#", low.color1)))){** *# check that all strings of low.color1 start by #*

**tempo.cat <- paste0("ERROR IN ", function.name, ": low.color1 ARGUMENT MUST BE A HEXADECIMAL COLOR VECTOR STARTING BY # AND/OR COLOR NAMES GIVEN BY colors()\n\n================\n\n")**

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

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

**}**

**if( ! is.null(mid.color1)){**

**tempo <- fun\_check(data = mid.color1, class = "character", length = 1, fun.name = function.name) ; eval(ee)**

**if(tempo$problem == FALSE & ! (all(mid.color1 %in% colors() | grepl(pattern = "^#", mid.color1)))){** *# check that all strings of mid.color1 start by #*

**tempo.cat <- paste0("ERROR IN ", function.name, ": mid.color1 ARGUMENT MUST BE A HEXADECIMAL COLOR VECTOR STARTING BY # AND/OR COLOR NAMES GIVEN BY colors()\n\n================\n\n")**

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

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

**}**

**}**

**tempo <- fun\_check(data = high.color1, class = "character", length = 1, fun.name = function.name) ; eval(ee)**

**if(tempo$problem == FALSE & ! (all(high.color1 %in% colors() | grepl(pattern = "^#", high.color1)))){** *# check that all strings of high.color1 start by #*

**tempo.cat <- paste0("ERROR IN ", function.name, ": high.color1 ARGUMENT MUST BE A HEXADECIMAL COLOR VECTOR STARTING BY # AND/OR COLOR NAMES GIVEN BY colors()\n\n================\n\n")**

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

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

**}**

**if( ! is.null(limit1)){**

**tempo <- fun\_check(data = limit1, class = "vector", mode = "numeric", length = 2, fun.name = function.name) ; eval(ee)**

**if(tempo$problem == FALSE & any(limit1 %in% c(Inf, -Inf))){**

**tempo.cat <- paste0("ERROR IN ", function.name, ": limit1 ARGUMENT CANNOT CONTAIN -Inf OR Inf VALUES\n\n================\n\n")**

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

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

**}**

**}**

**if( ! is.null(midpoint1)){**

**tempo <- fun\_check(data = midpoint1, class = "vector", mode = "numeric", length = 1, fun.name = function.name) ; eval(ee)**

**}**

**if( ! is.null(data2)){**

**if(all(is.matrix(data2))){**

**tempo <- fun\_check(data = data2, class = "matrix", mode = "numeric", fun.name = function.name) ; eval(ee)**

**if(tempo$problem == FALSE & ! all(unique(data2) %in% c(0,1))){**

**tempo.cat <- paste0("ERROR IN ", function.name, ": MATRIX IN data2 MUST BE MADE OF 0 AND 1 ONLY (MASK MATRIX)\n\n================\n\n")**

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

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

**}else if(tempo$problem == FALSE & all(is.matrix(data1)) & ! identical(dim(data1), dim(data2))){** *# matrix and matrix*

**tempo.cat <- paste0("ERROR IN ", function.name, ": MATRIX DIMENSION IN data2 MUST BE IDENTICAL AS MATRIX DIMENSION IN data1. HERE IT IS RESPECTIVELY:\n", paste(dim(data2), collapse = " "), "\n", paste(dim(data1), collapse = " "))**

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

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

**}else if(tempo$problem == FALSE & all(is.data.frame(data1)) & nrow(data1) != prod(dim(data2))){** *# reshape2 and matrix*

**tempo.cat <- paste0("ERROR IN ", function.name, ": DATA FRAME IN data2 MUST HAVE ROW NUMBER EQUAL TO PRODUCT OF DIMENSIONS OF data1 MATRIX. HERE IT IS RESPECTIVELY:\n", paste(nrow(data1), collapse = " "), "\n", paste(prod(dim(data2)), collapse = " "))**

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

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

**}**

**}else if(all(is.data.frame(data2))){**

**tempo <- fun\_check(data = data2, class = "data.frame", length = 3, fun.name = function.name) ; eval(ee)**

**if(tempo$problem == FALSE){**

*# structure of reshape2::melt() data frame*

**tempo <- fun\_check(data = data2[, 1], typeof = "integer", fun.name = function.name)**

**tempo <- fun\_check(data = data2[, 2], typeof = "integer", fun.name = function.name)**

**tempo <- fun\_check(data = data2[, 3], mode = "numeric", fun.name = function.name)**

**}**

**if(tempo$problem == FALSE & ! all(unique(data2[, 3]) %in% c(0,1))){**

**tempo.cat <- paste0("ERROR IN ", function.name, ": THIRD COLUMN OF DATA FRAME IN data2 MUST BE MADE OF 0 AND 1 ONLY (MASK DATA FRAME)\n\n================\n\n")**

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

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

**}else if(tempo$problem == FALSE & all(is.data.frame(data1)) & ! identical(dim(data1), dim(data2))){** *# data frame and data frame*

**tempo.cat <- paste0("ERROR IN ", function.name, ": DATA FRAME DIMENSION IN data2 MUST BE IDENTICAL TO DATA FRAME DIMENSION IN data1. HERE IT IS RESPECTIVELY:\n", paste(dim(data2), collapse = " "), "\n", paste(dim(data1), collapse = " "))**

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

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

**}else if(tempo$problem == FALSE & all(is.matrix(data1)) & nrow(data2) != prod(dim(data1))){** *# reshape2 and matrix*

**tempo.cat <- paste0("ERROR IN ", function.name, ": DATA FRAME IN data2 MUST HAVE ROW NUMBER EQUAL TO PRODUCT OF DIMENSION OF data1 MATRIX. HERE IT IS RESPECTIVELY:\n", paste(nrow(data2), collapse = " "), "\n", paste(prod(dim(data1)), collapse = " "))**

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

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

**}**

**}else{**

**tempo.cat <- paste0("ERROR IN ", function.name, ": THE data2 ARGUMENT MUST BE A NUMERIC MATRIX OR A DATA FRAME OUTPUT OF THE reshape::melt() FUNCTION\n\n================\n\n")**

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

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

**}**

**}**

**tempo <- fun\_check(data = color2, class = "character", length = 1, fun.name = function.name) ; eval(ee)**

**if(tempo$problem == FALSE & ! (all(color2 %in% colors() | grepl(pattern = "^#", color2)))){** *# check that all strings of color2 start by #*

**tempo.cat <- paste0("ERROR IN ", function.name, ": color2 ARGUMENT MUST BE A HEXADECIMAL COLOR VECTOR STARTING BY # AND/OR COLOR NAMES GIVEN BY colors()\n\n================\n\n")**

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

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

**}**

**tempo <- fun\_check(data = alpha2, class = "vector", mode = "numeric", length = 1, prop = TRUE, fun.name = function.name) ; eval(ee)**

**tempo <- fun\_check(data = invert2, class = "logical", length = 1, fun.name = function.name) ; eval(ee)**

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

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

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

**tempo <- fun\_check(data = show.scale, class = "logical", length = 1, 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){**

**if( ! all(dir.exists(lib.path))){** *# separation to avoid the problem of tempo$problem == FALSE and lib.path == NA*

**tempo.cat <- paste0("ERROR IN ", function.name, ": DIRECTORY PATH INDICATED IN THE lib.path ARGUMENT DOES NOT EXISTS:\n", paste(lib.path, collapse = "\n"))**

**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("reshape2", "ggplot2"), lib.path = lib.path)**

*# end package checking*

*# main code*

**if(all(is.matrix(data1))){**

**data1 <- reshape2::melt(data1)** *# transform a matrix into a dataframe with 2 coordinates columns and the third intensity column*

**}**

**if(rotate == TRUE){**

**data1[, 1] <- rev(data1[, 1])**

**}**

**if(is.null(limit1)){**

**if(any(data1[, 3] %in% c(Inf, -Inf))){**

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

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

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

**}**

**limit1 <- range(data1[, 3], 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*

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

**tempo.warn <- paste0("(", warn.count,") FROM FUNCTION ", function.name, ": THE limit1 ARGUMENT IS NULL -> RANGE OF data1 ARGUMENT HAS BEEN TAKEN: ", paste(fun\_round(limit1), collapse = " "))**

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

**if(suppressWarnings(any(limit1 %in% c(Inf, -Inf)))){**

**tempo.cat <- paste0("\n\n================\n\nERROR IN ", function.name, " COMPUTED LIMIT CONTAINS Inf VALUES, BECAUSE VALUES FROM data1 ARGUMENTS ARE NA OR Inf ONLY\n\n================\n\n")**

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

**}**

**}**

**if(is.null(midpoint1)){**

**midpoint1 <- mean(limit1, na.rm = TRUE)**

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

**tempo.warn <- paste0("(", warn.count,") FROM FUNCTION ", function.name, ": THE midpoint1 ARGUMENT IS NULL -> MEAN OF limit1 ARGUMENT HAS BEEN TAKEN: ", paste(fun\_round(midpoint1), collapse = " "))**

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

**}else if(fun\_round(midpoint1, 9) != fun\_round(mean(limit1), 9)){**

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

**tempo.warn <- paste0("(", warn.count,") FROM FUNCTION ", function.name, ": THE midpoint1 ARGUMENT (", fun\_round(mean(midpoint1), 9), ") DOES NOT CORRESPOND TO THE MEAN OF THE limit1 ARGUMENT (", fun\_round(mean(limit1), 9), "). COLOR SCALE IS NOT LINEAR")**

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

**}**

**if( ! is.null(data2)){**

**if(all(is.matrix(data2))){**

**data2 <- reshape2::melt(data2)** *# transform a matrix into a dataframe with 2 coordinates columns and the third intensity column*

**}**

**if(rotate == TRUE){**

**data2[, 1] <- rev(data2[, 1])**

**}**

**data2[, 3] <- factor(data2[, 3])** *# to converte continuous scale into discrete scale*

**}**

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

**tempo.gg.count <- 0** *# to facilitate debugging*

**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::geom\_raster(data = data1, mapping = ggplot2::aes\_string(x = names(data1)[ifelse(rotate == FALSE, 2, 1)], y = names(data1)[ifelse(rotate == FALSE, 1, 2)], fill = names(data1)[3]), show.legend = show.scale))** *# 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\_gradient2(low = low.color1, high = high.color1, mid = mid.color1, midpoint = midpoint1, limit = limit1, breaks = c(limit1[1], midpoint1, limit1[2]), labels = fun\_round(c(limit1[1], midpoint1, limit1[2])), name = legend.name1))**

**if( ! is.null(data2)){**

**assign(paste0(tempo.gg.name, tempo.gg.count <- tempo.gg.count + 1), ggplot2::geom\_raster(data = data2, mapping = ggplot2::aes\_string(x = names(data2)[ifelse(rotate == FALSE, 2, 1)], y = names(data2)[ifelse(rotate == FALSE, 1, 2)], alpha = names(data2)[3]), fill = color2, show.legend = FALSE))**

**assign(paste0(tempo.gg.name, tempo.gg.count <- tempo.gg.count + 1), ggplot2::scale\_discrete\_manual(aesthetics = "alpha", values = if(invert2 == FALSE){c(0, alpha2)}else{c(alpha2, 0)}, guide = FALSE))**

*# assign(paste0(tempo.gg.name, tempo.gg.count <- tempo.gg.count + 1), ggplot2::geom\_raster(data = data2, mapping = ggplot2::aes\_string(x = names(data2)[ifelse(rotate == FALSE, 2, 1)], y = names(data2)[ifelse(rotate == FALSE, 1, 2)], group = names(data2)[3]), fill = data2[, 3], alpha = alpha2, show.legend = FALSE)) # BEWARE: this does not work if NA present, because geom\_raster() has a tendency to complete empty spaces, and thus, behave differently than geom\_tile(). See https://github.com/tidyverse/ggplot2/issues/3025*

**}**

**assign(paste0(tempo.gg.name, tempo.gg.count <- tempo.gg.count + 1), ggplot2::coord\_fixed())** *# x = y*

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

**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")**

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

**add.check <- FALSE**

**}**

**}**

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

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

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

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

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

**line = ggplot2::element\_blank(),**

**axis.title = ggplot2::element\_blank(),**

**axis.text = ggplot2::element\_blank(),**

**axis.ticks = ggplot2::element\_blank(),**

**panel.background = ggplot2::element\_blank()**

**))**

**}**

**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)**

**}**

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

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

**output <- output$data**

**names(output)[1] <- "heatmap"**

**if( ! is.null(data2)){**

**names(output)[2] <- "mask"**

**}**

**return(list(data = output, axes = output$layout$panel\_params[[1]], scale = c(limit1[1], midpoint1, limit1[2]), warn = warn))**

**}**

**}**