**################################################################**

**## ##**

**## CUTE LITTLE R FUNCTIONS v6.0.0 ##**

**## ##**

**## Gael A. Millot ##**

**## ##**

**## Compatible with R v3.5.2 ##**

**## ##**

**################################################################**

**# BEWARE: do not forget to save the modifications in the .R file (through RSTUDIO for indentation)**

**################################ OUTLINE ################################**

[################ Object analysis 2](#_Toc17410972)

[######## fun\_param\_check() #### Checking class, type, length, etc. of objects 2](#_Toc17410973)

[######## fun\_object\_info() #### Recovering object information 8](#_Toc17410974)

[######## fun\_1D\_comp() #### comparison of two 1D datasets (vectors, factors, 1D tables) 9](#_Toc17410975)

[######## fun\_2D\_comp() #### comparison of two 2D datasets (row & col names, dimensions, etc.) 13](#_Toc17410976)

[######## fun\_2D\_head() #### head of the left or right of big 2D objects 19](#_Toc17410977)

[######## fun\_2D\_tail() #### tail of the left or right of big 2D objects 21](#_Toc17410978)

[######## fun\_list\_comp() #### comparison of two lists 22](#_Toc17410979)

[################ Object modification 24](#_Toc17410980)

[######## fun\_dataframe\_remodeling() #### remodeling a data frame to have column name as a qualitative column and vice-versa 24](#_Toc17410981)

[######## fun\_refactorization() #### remove classes that are not anymore present in factors or factor columns in data frames 27](#_Toc17410982)

[######## fun\_round() #### Rounding number if decimal present 29](#_Toc17410983)

[######## fun\_90clock\_matrix\_rot() #### 90° clockwise matrix rotation 31](#_Toc17410984)

[######## fun\_num2color\_mat() #### Conversion of a numeric matrix into hexadecimal color matrix 32](#_Toc17410985)

[######## fun\_by\_case\_matrix\_op() #### assembling of several matrices with operation 35](#_Toc17410986)

[######## fun\_mat\_inv() #### return the inverse of a square matrix 37](#_Toc17410987)

[######## fun\_mat\_fill() #### fill the empty half part of a symmetric square matrix 38](#_Toc17410988)

[######## fun\_consec\_pos\_perm() #### progressively breaks a vector order 41](#_Toc17410989)

[################ Graphics management 46](#_Toc17410990)

[######## fun\_window\_width\_resizing() #### window width depending on classes to plot 46](#_Toc17410991)

[######## fun\_open\_window() #### Open a GUI or pdf graphic window 47](#_Toc17410992)

[######## fun\_prior\_plot() #### Graph param before plotting 51](#_Toc17410993)

[######## fun\_post\_plot() #### Graph param after plotting 55](#_Toc17410994)

[######## fun\_close\_specif\_window() #### Closing specific graphic windows 66](#_Toc17410995)

[################ Standard graphics 67](#_Toc17410996)

[######## fun\_empty\_graph() #### text to display for empty graphs 67](#_Toc17410997)

[################ gg graphics 69](#_Toc17410998)

[######## fun\_gg\_palette() #### ggplot2 default color palette 69](#_Toc17410999)

[######## fun\_gg\_scatter() #### ggplot2 scatterplot + lines (up to 6 overlays totally) 70](#_Toc17411000)

[######## fun\_gg\_bar\_mean() #### ggplot2 mean barplot + overlaid dots if required 89](#_Toc17411001)

[######## fun\_gg\_boxplot() #### ggplot2 boxplot + background dots if required 110](#_Toc17411002)

[######## fun\_gg\_bar\_prop() #### ggplot2 proportion barplot 130](#_Toc17411003)

[######## fun\_gg\_strip() #### ggplot2 stripchart + mean/median 130](#_Toc17411004)

[######## fun\_gg\_violin() #### ggplot2 violins 130](#_Toc17411005)

[######## fun\_gg\_line() #### ggplot2 lines + background dots and error bars 130](#_Toc17411006)

[######## fun\_gg\_heatmap() #### ggplot2 heatmap + overlaid mask if required 130](#_Toc17411007)

[######## fun\_gg\_empty\_graph() #### text to display for empty graphs 135](#_Toc17411008)

[################ Graphic extraction 136](#_Toc17411009)

[######## fun\_var\_trim\_display() #### Display values from a quantitative variable and trim according to defined cut-offs 137](#_Toc17411010)

[######## fun\_segmentation() #### Segment a dot cloud on a scatterplot and define the dots from another cloud outside the segmentation 145](#_Toc17411011)

[################ Import 160](#_Toc17411012)

[######## fun\_pack\_import() #### Check if R packages are present and import into the working environment 160](#_Toc17411013)

[######## fun\_python\_pack\_import() #### Check if python packages are present 162](#_Toc17411014)

[################ Exporting results (text & tables) 163](#_Toc17411015)

[######## fun\_export\_data() #### Print string or data object into output file 163](#_Toc17411016)

**################################ FUNCTIONS ################################**

################ Object analysis

######## fun\_param\_check() #### Checking class, type, length, etc. of objects

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

**fun\_param\_check <- function(data, data.name = NULL, class = NULL, typeof = NULL, mode = NULL, length = NULL, prop = NULL, double.as.integer.allowed = FALSE, options = NULL, all.options.in.data = FALSE, na.contain = FALSE, neg.values = TRUE, print = TRUE, fun.name = NULL){**

*# AIM*

*# check the class, type, mode and length of the data argument*

*# mainly used to check the arguments of other functions*

*# check also other kind of data parameters, is it a proportion? Is it type double but numbers without decimal part?*

*# if options = NULL, then at least class, type, mode or length must be non null*

*# if options is non null, then class, type and mode must be NULL, and length can be NULL or specified*

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

*# none*

*# ARGUMENTS*

*# data: object to test*

*# data.name: name of the object to test. If NULL, use the name of the object assigned to the data argument*

*# class: one of the class() result or "vector"*

*# typeof: one of the typeof() result*

*# mode: one of the mode() result (for non vector object)*

*# length: length of the object*

*# prop: logical, are the numeric values between 0 and 1 (proportion)?*

*# double.as.integer.allowed: logical. If TRUE, no error is reported if argument is set to typeof = "integer" or class = "integer", while the reality is typeof = "double" or class = "numeric" but the numbers have a zero as modulo (remainder of a division). This means that i<-1 , which is typeof(i) -> "double" is considered as integer with double.as.integer.allowed = TRUE*

*# options: a vector of possible values for data*

*# all.options.in.data: If TRUE, all of the options must be present at least once in data, and nothing else. If FALSE, some of the options must be present in data, and nothing else*

*# na.contain: can data contains NA?*

*# neg.values: are negative numeric values authorized? BEWARE: only considered if set to FALSE, to check for non negative values when class is set to "numeric", "matrix", "array", "data.frame", "table", or typeof is set to "double", "integer", or mode is set to "numeric"*

*# print: print the error message if $problem is TRUE?*

*# fun.name: name of the function when fun\_param\_check() is used to check its argument. If non NULL, name will be added into the error message returned by fun\_param\_check()*

*# RETURN*

*# a list containing:*

*# $problem: logical. Is there any problem detected ?*

*# $text: the problem detected*

*# $param.name: name of the checked parameter*

*# EXAMPLES*

*# test <- 1:3 ; fun\_param\_check(data = test, data.name = NULL, print = TRUE, options = NULL, all.options.in.data = FALSE, class = NULL, typeof = NULL, mode = NULL, prop = TRUE, double.as.integer.allowed = FALSE, length = NULL)*

*# test <- 1:3 ; fun\_param\_check(data = test, print = TRUE, class = "numeric", typeof = NULL, double.as.integer.allowed = FALSE)*

*# test <- 1:3 ; fun\_param\_check(data = test, print = TRUE, class = "vector", mode = "numeric")*

*# test <- matrix(1:3) ; fun\_param\_check(data = test, print = TRUE, class = "vector", mode = "numeric")*

*# DEBUGGING*

*# data = 1:3 ; data.name = NULL ; print = TRUE; options = NULL ; all.options.in.data = FALSE ; class = "numeric" ; typeof = NULL ; mode = NULL ; prop = NULL ; double.as.integer.allowed = TRUE ; length = NULL # for function debugging*

*# argument checking*

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

**if( ! (length(data.name) == 1 & class(data.name) == "character")){**

**tempo.cat <- paste0("\n\n================\n\nERROR IN fun\_param\_check(): data.name ARGUMENT MUST BE A SINGLE CHARACTER ELEMENT AND NOT ", paste(data.name, collapse = " "), "\n\n================\n\n")**

**stop(tempo.cat)**

**}**

**}**

**if(is.null(options) & is.null(class) & is.null(typeof) & is.null(mode) & is.null(prop) & is.null(length)){**

**tempo.cat <- paste0("\n\n================\n\nERROR IN fun\_param\_check(): AT LEAST ONE OF THE options, class, typeof, mode, prop, OR length ARGUMENT MUST BE SPECIFIED\n\n================\n\n")**

**stop(tempo.cat)**

**}**

**if( ! is.null(options) & ( ! is.null(class) | ! is.null(typeof) | ! is.null(mode) | ! is.null(prop))){**

**tempo.cat <- paste0("\n\n================\n\nERROR IN fun\_param\_check(): THE class, typeof, mode AND prop ARGUMENTS MUST BE NULL IF THE option ARGUMENT IS SPECIFIED\nTHE option ARGUMENT MUST BE NULL IF THE class AND/OR typeof AND/OR mode AND/OR prop ARGUMENT IS SPECIFIED\n\n================\n\n")**

**stop(tempo.cat)**

**}**

**if( ! (all(class(neg.values) == "logical") & length(neg.values) == 1 & any(is.na(neg.values)) != TRUE)){**

**tempo.cat <- paste0("\n\n================\n\nERROR IN fun\_param\_check(): THE neg.values ARGUMENT MUST BE TRUE OR FALSE ONLY\n\n================\n\n")**

**stop(tempo.cat)**

**}**

**if(neg.values == FALSE & is.null(class) & is.null(typeof) & is.null(mode)){**

**tempo.cat <- paste0("\n\n================\n\nERROR IN fun\_param\_check(): THE neg.values ARGUMENT CANNOT BE SWITCHED TO FALSE IF class, typeof AND mode ARGUMENTS ARE NULL\n\n================\n\n")**

**stop(tempo.cat)**

**}**

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

**if( ! all(class %in% c("vector", "logical", "integer", "numeric", "complex", "character", "matrix", "array", "data.frame", "list", "factor", "table", "expression", "name", "symbol", "function") & any(is.na(class)) != TRUE)){** *# not length == 1 here because ordered factors are class "factor" "ordered" (length == 2)*

**tempo.cat <- paste0("\n\n================\n\nERROR IN fun\_param\_check(): class ARGUMENT MUST BE ONE OF THESE VALUE:\n\"vector\", \"logical\", \"integer\", \"numeric\", \"complex\", \"character\", \"matrix\", \"array\", \"data.frame\", \"list\", \"factor\", \"table\", \"expression\", \"name\", \"symbol\", \"function\" \n\n================\n\n")**

**stop(tempo.cat)**

**}**

**if(neg.values == FALSE & ! any(class %in% c("vector", "numeric", "integer", "matrix", "array", "data.frame", "table"))){**

**tempo.cat <- paste0("\n\n================\n\nERROR IN fun\_param\_check(): class ARGUMENT CANNOT BE OTHER THAN \"vector\", \"numeric\", \"integer\", \"matrix\", \"array\", \"data.frame\", \"table\" IF neg.values ARGUMENT IS SWITCHED TO FALSE\n\n================\n\n")**

**stop(tempo.cat)**

**}**

**}**

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

**if( ! (all(typeof %in% c("logical", "integer", "double", "complex", "character", "list", "expression", "name", "symbol", "closure", "special", "builtin")) & length(typeof) == 1 & any(is.na(typeof)) != TRUE)){**

**tempo.cat <- paste0("\n\n================\n\nERROR IN fun\_param\_check(): typeof ARGUMENT MUST BE ONE OF THESE VALUE:\n\"logical\", \"integer\", \"double\", \"complex\", \"character\", \"list\", \"expression\", \"name\", \"symbol\", \"closure\", \"special\", \"builtin\" \n\n================\n\n")**

**stop(tempo.cat)**

**}**

**if(neg.values == FALSE & ! typeof %in% c("double", "integer")){**

**tempo.cat <- paste0("\n\n================\n\nERROR IN fun\_param\_check(): typeof ARGUMENT CANNOT BE OTHER THAN \"double\" OR \"integer\" IF neg.values ARGUMENT IS SWITCHED TO FALSE\n\n================\n\n")**

**stop(tempo.cat)**

**}**

**}**

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

**if( ! (all(mode %in% c("logical", "numeric", "complex", "character", "list", "expression", "name", "symbol", "function")) & length(mode) == 1 & any(is.na(mode)) != TRUE)){**

**tempo.cat <- paste0("\n\n================\n\nERROR IN fun\_param\_check(): mode ARGUMENT MUST BE ONE OF THESE VALUE:\n\"logical\", \"numeric\", \"complex\", \"character\", \"list\", \"expression\", \"name\", \"symbol\", \"function\"\n\n================\n\n")**

**stop(tempo.cat)**

**}**

**if(neg.values == FALSE & mode != "numeric"){**

**tempo.cat <- paste0("\n\n================\n\nERROR IN fun\_param\_check(): mode ARGUMENT CANNOT BE OTHER THAN \"numeric\" IF neg.values ARGUMENT IS SWITCHED TO FALSE\n\n================\n\n")**

**stop(tempo.cat)**

**}**

**}**

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

**if( ! (is.numeric(length) & length(length) == 1 & ! grepl(length, pattern = "\\.") & any(is.na(length)) != TRUE)){**

**tempo.cat <- paste0("\n\n================\n\nERROR IN fun\_param\_check(): length ARGUMENT MUST BE A SINGLE INTEGER VALUE\n\n================\n\n")**

**stop(tempo.cat)**

**}**

**}**

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

**if( ! (is.logical(prop) | length(prop) == 1 & any(is.na(prop)) != TRUE)){**

**tempo.cat <- paste0("\n\n================\n\nERROR IN fun\_param\_check(): prop ARGUMENT MUST BE TRUE OR FALSE ONLY\n\n================\n\n")**

**stop(tempo.cat)**

**}else if(prop == TRUE){**

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

**if( ! any(class %in% c("vector", "numeric", "integer", "matrix", "array", "data.frame", "table"))){**

**tempo.cat <- paste0("\n\n================\n\nERROR IN fun\_param\_check(): class ARGUMENT CANNOT BE OTHER THAN \"vector\", \"numeric\", \"integer\", \"matrix\", \"array\", \"data.frame\", \"table\" IF prop ARGUMENT IS TRUE\n\n================\n\n")**

**stop(tempo.cat)**

**}**

**}**

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

**if(mode != "numeric"){**

**tempo.cat <- paste0("\n\n================\n\nERROR IN fun\_param\_check(): mode ARGUMENT CANNOT BE OTHER THAN \"numeric\" IF prop ARGUMENT IS TRUE\n\n================\n\n")**

**stop(tempo.cat)**

**}**

**}**

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

**if(typeof != "double"){**

**tempo.cat <- paste0("\n\n================\n\nERROR IN fun\_param\_check(): typeof ARGUMENT CANNOT BE OTHER THAN \"double\" IF prop ARGUMENT IS TRUE\n\n================\n\n")**

**stop(tempo.cat)**

**}**

**}**

**}**

**}**

**if( ! (all(class(double.as.integer.allowed) == "logical") & length(double.as.integer.allowed) == 1 & any(is.na(double.as.integer.allowed)) != TRUE)){**

**tempo.cat <- paste0("\n\n================\n\nERROR IN fun\_param\_check(): THE double.as.integer.allowed ARGUMENT MUST BE TRUE OR FALSE ONLY\n\n================\n\n")**

**stop(tempo.cat)**

**}**

**if( ! (is.logical(all.options.in.data) & length(all.options.in.data) == 1 & any(is.na(all.options.in.data)) != TRUE)){**

**tempo.cat <- paste0("\n\n================\n\nERROR IN fun\_param\_check(): all.options.in.data ARGUMENT MUST BE A SINGLE LOGICAL VALUE (TRUE OR FALSE ONLY)\n\n================\n\n")**

**stop(tempo.cat)**

**}**

**if( ! (all(class(na.contain) == "logical") & length(na.contain) == 1 & any(is.na(na.contain)) != TRUE)){**

**tempo.cat <- paste0("\n\n================\n\nERROR IN fun\_param\_check(): THE na.contain ARGUMENT MUST BE TRUE OR FALSE ONLY\n\n================\n\n")**

**stop(tempo.cat)**

**}**

**if( ! (all(class(print) == "logical") & length(print) == 1 & any(is.na(print)) != TRUE)){**

**tempo.cat <- paste0("\n\n================\n\nERROR IN fun\_param\_check(): THE print ARGUMENT MUST BE TRUE OR FALSE ONLY\n\n================\n\n")**

**stop(tempo.cat)**

**}**

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

**if( ! (class(fun.name) == "character" & length(fun.name) == 1)){**

**tempo.cat <- paste0("\n\n================\n\nERROR IN fun\_param\_check(): THE fun.name ARGUMENT MUST BE A CHARACTER VECTOR OF LENGTH 1\n\n================\n\n")**

**stop(tempo.cat)**

**}**

**}**

*# 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)) # activate this line and use the function to check arguments status*

*# end argument checking*

*# main code*

**if(is.null(data.name)){**

**data.name <- deparse(substitute(data))**

**}**

**problem <- FALSE**

**text <- paste0(ifelse(is.null(fun.name), "", paste0("IN ", fun.name, ": ")), "NO PROBLEM DETECTED FOR THE ", data.name, " PARAMETER")**

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

**text <- ""**

**if( ! all(data %in% options)){**

**problem <- TRUE**

**text <- paste0(ifelse(is.null(fun.name), "ERROR", paste0("ERROR IN ", fun.name)), ": THE ", data.name, " PARAMETER MUST BE SOME OF THESE OPTIONS: ", paste(options, collapse = " "), "\nTHE PROBLEMATIC ELEMENTS OF ", data.name, " ARE: ", paste(unique(data[ ! (data %in% options)]), collapse = " "))**

**}**

**if(all.options.in.data == TRUE){**

**if( ! all(options %in% data)){**

**problem <- TRUE**

**if(text == ""){**

**text <- paste0(ifelse(is.null(fun.name), "ERROR", paste0("ERROR IN ", fun.name)), ": THE ", data.name, " PARAMETER MUST BE SOME OF THESE OPTIONS: ", paste(options, collapse = " "), "\nTHE PROBLEMATIC ELEMENTS OF ", data.name, " ARE: ", unique(data[ ! (data %in% options)]))**

**}else{**

**text <- paste0(text, "\n", ifelse(is.null(fun.name), "ERROR", paste0("ERROR IN ", fun.name)), ": THE ", data.name, " PARAMETER MUST BE SOME OF THESE OPTIONS: ", paste(options, collapse = " "), "\nTHE PROBLEMATIC ELEMENTS OF ", data.name, " ARE: ", unique(data[ ! (data %in% options)]))**

**}**

**}**

**}**

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

**if(length(data) != length){**

**problem <- TRUE**

**if(text == ""){**

**text <- paste0(ifelse(is.null(fun.name), "ERROR", paste0("ERROR IN ", fun.name)), ": THE LENGTH OF ", data.name, " MUST BE ", length, " AND NOT ", length(data))**

**}else{**

**text <- paste0(text, "\n", ifelse(is.null(fun.name), "ERROR", paste0("ERROR IN ", fun.name)), ": THE LENGTH OF ", data.name, " MUST BE ", length, " AND NOT ", length(data))**

**}**

**}**

**}**

**if(text == ""){**

**text <- paste0(ifelse(is.null(fun.name), "", paste0("IN ", fun.name, ": ")), "NO PROBLEM DETECTED FOR THE ", data.name, " PARAMETER")**

**}**

**}**

**arg.names <- c("class", "typeof", "mode", "length")**

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

**for(i2 in 1:length(arg.names)){**

**if( ! is.null(get(arg.names[i2]))){**

*# script to execute*

**tempo.script <- '**

**problem <- TRUE ;**

**if(identical(text, paste0(ifelse(is.null(fun.name), "", paste0("IN ", fun.name, ": ")), "NO PROBLEM DETECTED FOR THE ", data.name, " PARAMETER"))){**

**text <- paste0(ifelse(is.null(fun.name), "ERROR", paste0("ERROR IN ", fun.name)), ": THE ", data.name, " PARAMETER MUST BE ") ;**

**}else{**

**text <- paste0(text, " AND ");**

**}**

**text <- paste0(text, toupper(arg.names[i2]), " ", get(arg.names[i2]))**

**'**

*# end script to execute*

**if(typeof(data) == "double" & double.as.integer.allowed == TRUE & ((arg.names[i2] == "class" & get(arg.names[i2]) == "integer") | (arg.names[i2] == "typeof" & get(arg.names[i2]) == "integer"))){**

**if(! all(data%%1 == 0)){** *# to check integers (use %%, meaning the remaining of a division): see the precedent line*

**eval(parse(text = tempo.script))** *# execute tempo.script*

**}**

**}else if(get(arg.names[i2]) != "vector" & eval(parse(text = paste0(arg.names[i2], "(data)"))) != get(arg.names[i2])){**

**eval(parse(text = tempo.script))** *# execute tempo.script*

**}else if(arg.names[i2] == "class" & get(arg.names[i2]) == "vector" & ! (class(data) == "numeric" | class(data) == "integer" | class(data) == "character" | class(data) == "logical")){**

**eval(parse(text = tempo.script))** *# execute tempo.script*

**}**

**}**

**}**

**}**

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

**if(prop == TRUE){**

**if(any(data < 0 | data > 1, na.rm = TRUE)){**

**problem <- TRUE**

**if(identical(text, paste0(ifelse(is.null(fun.name), "", paste0("IN ", fun.name, ": ")), "NO PROBLEM DETECTED FOR THE ", data.name, " PARAMETER"))){**

**text <- paste0(ifelse(is.null(fun.name), "ERROR", paste0("ERROR IN ", fun.name)), ": ")**

**}else{**

**text <- paste0(text, " AND ")**

**}**

**text <- paste0(text, "THE ", data.name, " PARAMETER MUST BE DECIMAL VALUES BETWEEN 0 AND 1")**

**}**

**}**

**}**

**if(na.contain == FALSE & any(is.na(data)) == TRUE){**

**problem <- TRUE**

**if(identical(text, paste0(ifelse(is.null(fun.name), "", paste0("IN ", fun.name, ": ")), "NO PROBLEM DETECTED FOR THE ", data.name, " PARAMETER"))){**

**text <- paste0(ifelse(is.null(fun.name), "ERROR", paste0("ERROR IN ", fun.name)), ": ")**

**}else{**

**text <- paste0(text, " AND ")**

**}**

**text <- paste0(text, "THE ", data.name, " PARAMETER CONTAINS NA WHILE NOT AUTHORIZED (na.contain ARGUMENT SET TO FALSE)")**

**}**

**if(neg.values == FALSE){**

**if(any(data < 0, na.rm = TRUE)){**

**problem <- TRUE**

**if(identical(text, paste0(ifelse(is.null(fun.name), "", paste0("IN ", fun.name, ": ")), "NO PROBLEM DETECTED FOR THE ", data.name, " PARAMETER"))){**

**text <- paste0(ifelse(is.null(fun.name), "ERROR", paste0("ERROR IN ", fun.name)), ": ")**

**}else{**

**text <- paste0(text, " AND ")**

**}**

**text <- paste0(text, "THE ", data.name, " PARAMETER MUST BE NON NEGATIVE NUMERIC VALUES")**

**}**

**}**

**if(print == TRUE & problem == TRUE){**

**cat(paste0("\n\n================\n\n", text, "\n\n================\n\n"))**

**}**

**output <- list(problem = problem, text = text, param.name = data.name)**

**return(output)**

**}**

######## fun\_object\_info() #### Recovering object information

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

**fun\_object\_info <- function(data){**

*# AIM*

*# provide a full description of the object*

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

*# none*

*# ARGUMENTS*

*# data: object to test*

*# RETURN*

*# a list containing the info*

*# use names(fun\_object\_info()) and remove what can be to big for easy analysis*

*# EXAMPLES*

*# fun\_object\_info(data = 1:3)*

*# fun\_object\_info(data.frame(a = 1:2, b = ordered(factor(c("A", "B")))))*

*# fun\_object\_info(list(a = 1:3, b = ordered(factor(c("A", "B")))))*

*# DEBUGGING*

*# data = NULL # for function debugging*

*# data = 1:3 # for function debugging*

*# data = matrix(1:3) # for function debugging*

*# data = data.frame(a = 1:2, b = c("A", "B")) # for function debugging*

*# data = factor(c("b", "a")) # for function debugging*

*# data = ordered(factor(c("b", "a"))) # for function debugging*

*# data = list(a = 1:3, b = factor(c("A", "B"))) # for function debugging*

*# data = list(a = 1:3, b = ordered(factor(c("A", "B")))) # for function debugging*

*# argument checking*

*# 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)) # activate this line and use the function to check arguments status*

*# end argument checking*

*# main code*

**data.name <- deparse(substitute(data))**

**output <- list("FILE\_NAME" = data.name)**

**tempo <- list("CLASS" = class(data))**

**output <- c(output, tempo)**

**tempo <- list("FILE\_HEAD" = head(data))**

**output <- c(output, tempo)**

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

**tempo <- list("FILE\_TAIL" = tail(data))**

**output <- c(output, tempo)**

**if( ! is.null(dim(data))){**

**tempo <- list("FILE\_DIMENSION" = dim(data))**

**names(tempo[[1]]) <- c("NROW", "NCOL")**

**output <- c(output, tempo)**

**}**

**tempo <- list("SUMMARY" = summary(data))**

**output <- c(output, tempo)**

**}**

**if(all(class(data) == "data.frame" | class(data) == "matrix")){**

**tempo <- list("ROW\_NAMES" = dimnames(data)[[1]])**

**output <- c(output, tempo)**

**tempo <- list("COLUM\_NAMES" = dimnames(data)[[2]])**

**output <- c(output, tempo)**

**}**

**if(all(class(data) == "data.frame")){**

**tempo <- list("STRUCTURE" = ls.str(data))**

**output <- c(output, tempo)**

**tempo <- list("COLUMN\_TYPE" = sapply(data, FUN = "typeof"))**

**if(any(sapply(data, FUN = "class") %in% "factor")){** *# if an ordered factor is present, then sapply(data, FUN = "class") return a list but works with any(sapply(data, FUN = "class") %in% "factor")*

**tempo.class <- sapply(data, FUN = "class")**

**if(any(unlist(tempo.class) %in% "ordered")){**

**tempo2 <- sapply(tempo.class, paste, collapse = " ")** *# paste the "ordered" factor" in "ordered factor"*

**}else{**

**tempo2 <- unlist(tempo.class)**

**}**

**tempo[["COLUMN\_TYPE"]][grepl(x = tempo2, pattern = "factor")] <- tempo2[grepl(x = tempo2, pattern = "factor")]**

**}**

**output <- c(output, tempo)**

**}**

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

**tempo <- list("COMPARTMENT\_NAMES" = names(data))**

**output <- c(output, tempo)**

**tempo <- list("COMPARTMENT\_TYPE" = sapply(data, FUN = "typeof"))**

**if(any(unlist(sapply(data, FUN = "class")) %in% "factor")){** *# if an ordered factor is present, then sapply(data, FUN = "class") return a list but works with any(sapply(data, FUN = "class") %in% "factor")*

**tempo.class <- sapply(data, FUN = "class")**

**if(any(unlist(tempo.class) %in% "ordered")){**

**tempo2 <- sapply(tempo.class, paste, collapse = " ")** *# paste the "ordered" factor" in "ordered factor"*

**}else{**

**tempo2 <- unlist(tempo.class)**

**}**

**tempo[["COMPARTMENT\_TYPE"]][grepl(x = tempo2, pattern = "factor")] <- tempo2[grepl(x = tempo2, pattern = "factor")]**

**}**

**output <- c(output, tempo)**

**}**

**return(output)**

**}**

######## fun\_1D\_comp() #### comparison of two 1D datasets (vectors, factors, 1D tables)

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

**fun\_1D\_comp <- function(data1, data2){**

*# AIM*

*# compare two 1D datasets (vector of factor or 1D table) of the same class or not. Check and report in a list if the 2 datasets have:*

*# same class*

*# common elements*

*# common element names (except factors)*

*# common levels (factors only)*

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

*# none*

*# ARGUMENTS*

*# data1: vector or factor or 1D table*

*# data2: vector or factor or 1D table*

*# RETURN*

*# a list containing:*

*# $same.class: logical. Are class identical?*

*# $class: class of the 2 datasets (NULL otherwise)*

*# $same.length: logical. Are number of elements identical?*

*# $length: number of elements in the 2 datasets (NULL otherwise)*

*# $same.levels: logical. Are levels identical? (NULL if data1 and data2 are not factors)*

*# $levels: levels of the 2 datasets if identical (NULL otherwise or NULL if data1 and data2 are not factors)*

*# $any.id.levels: logical. Is there any identical levels? (NULL if data1 and data2 are not factors)*

*# $same.levels.pos1: position, in data1, of the levels identical in data2 (NULL if data1 and data2 are not factors)*

*# $same.levels.pos2: position, in data2, of the levels identical in data1 (NULL if data1 and data2 are not factors)*

*# $common.levels: common levels between data1 and data2 (can be a subset of $levels or not). NULL if no common levels or if data1 and data2 are not factors*

*# $same.name: logical. Are element names identical ?*

*# $name: name of elements of the 2 datasets if identical (NULL otherwise)*

*# $any.id.name: logical. Is there any element names identical ?*

*# $same.name.pos1: position, in data1, of the element names identical in data2*

*# $same.name.pos2: position, in data2, of the elements names identical in data1*

*# $common.names: common element names between data1 and data2 (can be a subset of $name or not). NULL if no common element names*

*# $any.id.element: logical. is there any identical elements ?*

*# $same.element.pos1: position, in data1, of the elements identical in data2*

*# $same.element.pos2: position, in data2, of the elements identical in data1*

*# $common.elements: common elements between data1 and data2. NULL if no common elements*

*# $identical.object: logical. Are objects identical (kind of object, element names and content)?*

*# $identical.content: logical. Are content objects identical (identical elements excluding kind of object and element names)?*

*# EXAMPLES*

*# obs1 = 1:5 ; obs2 = 1:5 ; names(obs1) <- LETTERS[1:5] ; names(obs2) <- LETTERS[1:5] ; fun\_1D\_comp(obs1, obs2)*

*# obs1 = 1:5 ; obs2 = 1:5 ; names(obs1) <- LETTERS[1:5] ; fun\_1D\_comp(obs1, obs2)*

*# obs1 = 1:5 ; obs2 = 3:6 ; names(obs1) <- LETTERS[1:5] ; names(obs2) <- LETTERS[1:4] ; fun\_1D\_comp(obs1, obs2)*

*# obs1 = factor(LETTERS[1:5]) ; obs2 = factor(LETTERS[1:5]) ; fun\_1D\_comp(obs1, obs2)*

*# obs1 = factor(LETTERS[1:5]) ; obs2 = factor(LETTERS[10:11]) ; fun\_1D\_comp(obs1, obs2)*

*# obs1 = factor(LETTERS[1:5]) ; obs2 = factor(LETTERS[4:7]) ; fun\_1D\_comp(obs1, obs2)*

*# obs1 = 1:5 ; obs2 = factor(LETTERS[1:5]) ; fun\_1D\_comp(obs1, obs2)*

*# obs1 = 1:5 ; obs2 = 1.1:6.1 ; fun\_1D\_comp(obs1, obs2)*

*# obs1 = as.table(1:5); obs2 = as.table(1:5) ; fun\_1D\_comp(obs1, obs2)*

*# obs1 = as.table(1:5); obs2 = 1:5 ; fun\_1D\_comp(obs1, obs2)*

*# DEBUGGING*

*# data1 = 1:5 ; data2 = 1:5 ; names(data1) <- LETTERS[1:5] ; names(data2) <- LETTERS[1:5] # for function debugging*

*# function name*

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

*# end function name*

*# argument checking*

**if( ! any(class(data1) %in% c("logical", "integer", "numeric", "character", "factor", "table"))){**

**tempo.cat <- paste0("\n\n================\n\nERROR IN ", function.name, ": THE data1 ARGUMENT MUST BE A NON NULL VECTOR, FACTOR OR 1D TABLE\n\n================\n\n")**

**stop(tempo.cat)**

**}else if(all(class(data1) %in% "table")){**

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

**tempo.cat <- paste0("\n\n================\n\nERROR IN ", function.name, ": THE data1 ARGUMENT MUST BE A 1D TABLE\n\n================\n\n")**

**stop(tempo.cat)**

**}**

**}**

**if( ! any(class(data2) %in% c("logical", "integer", "numeric", "character", "factor", "table"))){**

**tempo.cat <- paste0("\n\n================\n\nERROR IN ", function.name, ": THE data2 ARGUMENT MUST BE A NON NULL VECTOR, FACTOR OR 1D TABLE\n\n================\n\n")**

**stop(tempo.cat)**

**}else if(all(class(data2) %in% "table")){**

**if(length(dim(data2)) > 1){**

**tempo.cat <- paste0("\n\n================\n\nERROR IN ", function.name, ": THE data2 ARGUMENT MUST BE A 1D TABLE\n\n================\n\n")**

**stop(tempo.cat)**

**}**

**}**

*# 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)) # activate this line and use the function to check arguments status*

*# end argument checking*

*# main code*

**same.class <- NULL**

**class <- NULL**

**same.length <- NULL**

**length <- NULL**

**same.levels <- NULL**

**levels <- NULL**

**any.id.levels <- NULL**

**same.levels.pos1 <- NULL**

**same.levels.pos2 <- NULL**

**common.levels <- NULL**

**same.name <- NULL**

**name <- NULL**

**any.id.name <- NULL**

**same.name.pos1 <- NULL**

**same.name.pos2 <- NULL**

**common.names <- NULL**

**any.id.element <- NULL**

**same.element.pos1 <- NULL**

**same.element.pos2 <- NULL**

**common.elements <- NULL**

**identical.object <- NULL**

**identical.content <- NULL**

**if(identical(data1, data2)){**

**same.class <- TRUE**

**class <- class(data1)**

**same.length <- TRUE**

**length <- length(data1)**

**if(any(class(data1) %in% "factor")){**

**same.levels <- TRUE**

**levels <- levels(data1)**

**any.id.levels <- TRUE**

**same.levels.pos1 <- 1:length(levels(data1))**

**same.levels.pos2 <- 1:length(levels(data2))**

**common.levels <- levels(data1)**

**}**

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

**same.name <- TRUE**

**name <- names(data1)**

**any.id.name <- TRUE**

**same.name.pos1 <- 1:length(data1)**

**same.name.pos2 <- 1:length(data2)**

**common.names <- names(data1)**

**}**

**any.id.element <- TRUE**

**same.element.pos1 <- 1:length(data1)**

**same.element.pos2 <- 1:length(data2)**

**common.elements <- data1**

**identical.object <- TRUE**

**identical.content <- TRUE**

**}else{**

**identical.object <- FALSE**

**if( ! identical(class(data1), class(data2))){**

**same.class <- FALSE**

**}else{**

**same.class <- TRUE**

**class <- class(data1)**

**}**

**if( ! identical(length(data1), length(data2))){**

**same.length<- FALSE**

**}else{**

**same.length<- TRUE**

**length <- length(data1)**

**}**

**if(any(class(data1) %in% "factor") & any(class(data2) %in% "factor")){**

**if( ! identical(levels(data1), levels(data2))){**

**same.levels <- FALSE**

**}else{**

**same.levels <- TRUE**

**levels <- levels(data1)**

**}**

**any.id.levels <- FALSE**

**if(any(levels(data1) %in% levels(data2))){**

**any.id.levels <- TRUE**

**same.levels.pos1 <- which(levels(data1) %in% levels(data2))**

**}**

**if(any(levels(data2) %in% levels(data1))){**

**any.id.levels <- TRUE**

**same.levels.pos2 <- which(levels(data2) %in% levels(data1))**

**}**

**if(any.id.levels == TRUE){**

**common.levels <- unique(c(levels(data1)[same.levels.pos1], levels(data2)[same.levels.pos2]))**

**}**

**}**

**if(any(class(data1) %in% "factor")){** *# to compare content*

**data1 <- as.character(data1)**

**}**

**if(any(class(data2) %in% "factor")){** *# to compare content*

**data2 <- as.character(data2)**

**}**

**if( ! (is.null(names(data1)) & is.null(names(data2)))){**

**if( ! identical(names(data1), names(data2))){**

**same.name <- FALSE**

**}else{**

**same.name <- TRUE**

**name <- names(data1)**

**}**

**any.id.name <- FALSE**

**if(any(names(data1) %in% names(data2))){**

**any.id.name <- TRUE**

**same.name.pos1 <- which(names(data1) %in% names(data2))**

**}**

**if(any(names(data2) %in% names(data1))){**

**any.id.name <- TRUE**

**same.name.pos2 <- which(names(data2) %in% names(data1))**

**}**

**if(any.id.name == TRUE){**

**common.names <- unique(c(names(data1)[same.name.pos1], names(data2)[same.name.pos2]))**

**}**

**}**

**any.id.element <- FALSE**

**if(any(data1 %in% data2)){**

**any.id.element <- TRUE**

**same.element.pos1 <- which(data1 %in% data2)**

**}**

**if(any(data2 %in% data1)){**

**any.id.element <- TRUE**

**same.element.pos2 <- which(data2 %in% data1)**

**}**

**if(any.id.element == TRUE){**

**common.elements <- unique(c(data1[same.element.pos1], data2[same.element.pos2]))**

**}**

**if(same.length == TRUE & ! all(is.null(same.element.pos1), is.null(same.element.pos2))){**

**names(same.element.pos1) <- NULL**

**names(same.element.pos2) <- NULL**

**if(identical(same.element.pos1, same.element.pos2)){**

**identical.content <- TRUE**

**}else{**

**identical.content <- FALSE**

**}**

**}else{**

**identical.content <- FALSE**

**}**

**}**

**output <- list(same.class = same.class, class = class, same.length = same.length, length = length, same.levels = same.levels, levels = levels, any.id.levels = any.id.levels, same.levels.pos1 = same.levels.pos1, same.levels.pos2 = same.levels.pos2, common.levels = common.levels, same.name = same.name, name = name, any.id.name = any.id.name, same.name.pos1 = same.name.pos1, same.name.pos2 = same.name.pos2, common.names = common.names, any.id.element = any.id.element, same.element.pos1 = same.element.pos1, same.element.pos2 = same.element.pos2, common.elements = common.elements, identical.object = identical.object, identical.content = identical.content)**

**return(output)**

**}**

######## fun\_2D\_comp() #### comparison of two 2D datasets (row & col names, dimensions, etc.)

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

**fun\_2D\_comp <- function(data1, data2){**

*# AIM*

*# compare two 2D datasets of the same class or not. Check and report in a list if the 2 datasets have:*

*# same class*

*# common row names*

*# common column names*

*# same row number*

*# same column number*

*# potential identical rows between the 2 datasets*

*# potential identical columns between the 2 datasets*

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

*# none*

*# ARGUMENTS*

*# data1: matrix, data frame or table*

*# data2: matrix, data frame or table*

*# RETURN*

*# a list containing:*

*# $same.class: logical. Are class identical ?*

*# $class: classes of the 2 datasets (NULL otherwise)*

*# $same.dim: logical. Are dimension identical ?*

*# $dim: dimension of the 2 datasets (NULL otherwise)*

*# $same.row.nb: logical. Are number of rows identical ?*

*# $row.nb: nb of rows of the 2 datasets if identical (NULL otherwise)*

*# $same.col.nb: logical. Are number of columns identical ?*

*# $col.nb: nb of columns of the 2 datasets if identical (NULL otherwise)*

*# $same.row.name: logical. Are row names identical ? NULL if no row names in the two 2D datasets*

*# $row.name: name of rows of the 2 datasets if identical (NULL otherwise)*

*# $any.id.row.name: logical. Is there any row names identical ? NULL if no row names in the two 2D datasets*

*# $same.row.name.pos1: position, in data1, of the row names identical in data2*

*# $same.row.name.pos2: position, in data2, of the row names identical in data1*

*# $common.row.names: common row names between data1 and data2 (can be a subset of $name or not). NULL if no common row names*

*# $same.col.name: logical. Are column names identical ? NULL if no col names in the two 2D datasets*

*# $col.name: name of columns of the 2 datasets if identical (NULL otherwise)*

*# $any.id.col.name: logical. Is there any column names identical ? NULL if no col names in the two 2D datasets*

*# $same.col.name.pos1: position, in data1, of the column names identical in data2*

*# $same.col.name.pos2: position, in data2, of the column names identical in data1*

*# $common.col.names: common column names between data1 and data2 (can be a subset of $name or not). NULL if no common column names*

*# $any.id.row: logical. is there identical rows (not considering row names) ?*

*# $same.row.pos1: position, in data1, of the rows identical in data2 (not considering row names)*

*# $same.row.pos2: position, in data2, of the rows identical in data1 (not considering row names)*

*# $any.id.col: logical. is there identical columns (not considering column names)?*

*# $same.col.pos1: position in data1 of the cols identical in data2 (not considering column names)*

*# $same.col.pos2: position in data2 of the cols identical in data1 (not considering column names)*

*# $identical.object: logical. Are objects identical (including row & column names)?*

*# $identical.content: logical. Are content objects identical (identical excluding row & column names)?*

*# EXAMPLES*

*# obs1 = matrix(1:10, ncol = 5, dimnames = list(letters[1:2], LETTERS[1:5])) ; obs2 = as.data.frame(matrix(1:10, ncol = 5, dimnames = list(letters[1:2], LETTERS[1:5]))) ; obs1 ; obs2 ; fun\_2D\_comp(obs1, obs2)*

*# obs1 = matrix(101:110, ncol = 5, dimnames = list(letters[1:2], LETTERS[1:5])) ; obs2 = matrix(1:10, ncol = 5, dimnames = list(letters[1:2], LETTERS[1:5])) ; obs1 ; obs2 ; fun\_2D\_comp(obs1, obs2)*

*# obs1 = matrix(1:10, byrow = TRUE, ncol = 5, dimnames = list(letters[1:2], LETTERS[1:5])) ; obs2 = matrix(c(1:5, 101:105, 6:10), byrow = TRUE, ncol = 5, dimnames = list(c("a", "z", "b"), c(LETTERS[1:2], "k", LETTERS[5:4]))) ; obs1 ; obs2 ; fun\_2D\_comp(obs1, obs2)*

*# obs1 = t(matrix(1:10, byrow = TRUE, ncol = 5, dimnames = list(letters[1:2], LETTERS[1:5]))) ; obs2 = t(matrix(c(1:5, 101:105, 6:10), byrow = TRUE, ncol = 5, dimnames = list(c("a", "z", "b"), c(LETTERS[1:2], "k", LETTERS[5:4])))) ; obs1 ; obs2 ; fun\_2D\_comp(obs1, obs2)*

*# DEBUGGING*

*# data1 = matrix(1:10, ncol = 5) ; data2 = matrix(1:10, ncol = 5) # for function debugging*

*# data1 = matrix(1:10, ncol = 5, dimnames = list(letters[1:2], LETTERS[1:5])) ; data2 = matrix(1:10, ncol = 5, dimnames = list(letters[1:2], LETTERS[1:5])) # for function debugging*

*# data1 = matrix(1:10, ncol = 5, dimnames = list(letters[1:2], LETTERS[1:5])) ; data2 = matrix(1:10, ncol = 5) # for function debugging*

*# data1 = matrix(1:15, byrow = TRUE, ncol = 5, dimnames = list(letters[1:3], LETTERS[1:5])) ; data2 = matrix(1:10, byrow = TRUE, ncol = 5, dimnames = list(letters[1:2], LETTERS[1:5])) # for function debugging*

*# data1 = matrix(1:15, ncol = 5, dimnames = list(letters[1:3], LETTERS[1:5])) ; data2 = matrix(1:10, ncol = 5, dimnames = list(letters[1:2], LETTERS[1:5])) # for function debugging*

*# data1 = matrix(1:15, ncol = 5, dimnames = list(paste0("A", letters[1:3]), LETTERS[1:5])) ; data2 = matrix(1:10, ncol = 5, dimnames = list(letters[1:2], LETTERS[1:5])) # for function debugging*

*# data1 = matrix(1:15, ncol = 5, dimnames = list(letters[1:3], LETTERS[1:5])) ; data2 = matrix(1:12, ncol = 4, dimnames = list(letters[1:3], LETTERS[1:4])) # for function debugging*

*# data1 = matrix(1:10, ncol = 5, dimnames = list(letters[1:2], LETTERS[1:5])) ; data2 = matrix(101:110, ncol = 5, dimnames = list(letters[1:2], LETTERS[1:5])) # for function debugging*

*# data1 = data.frame(a = 1:3, b= letters[1:3], row.names = LETTERS[1:3]) ; data2 = data.frame(A = 1:3, B= letters[1:3]) # for function debugging*

*# data1 = matrix(1:10, ncol = 5, dimnames = list(letters[1:2], LETTERS[1:5])) ; data2 = as.data.frame(matrix(1:10, ncol = 5, dimnames = list(letters[1:2], LETTERS[1:5]))) # for function debugging*

*# data1 = matrix(1:10, byrow = TRUE, ncol = 5, dimnames = list(letters[1:2], LETTERS[1:5])) ; data2 = matrix(c(1:5, 101:105, 6:10), byrow = TRUE, ncol = 5, dimnames = list(c("a", "z", "b"), c(LETTERS[1:2], "k", LETTERS[5:4]))) # for function debugging*

*# data1 = table(Exp1 = c("A", "A", "A", "B", "B", "B"), Exp2 = c("A1", "B1", "A1", "C1", "C1", "B1")) ; data2 = data.frame(A = 1:3, B= letters[1:3]) # for function debugging*

*# function name*

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

*# end function name*

*# argument checking*

**if( ! any(class(data1) %in% c("matrix", "data.frame", "table"))){**

**tempo.cat <- paste0("\n\n================\n\nERROR IN ", function.name, ": THE data1 ARGUMENT MUST BE A MATRIX, DATA FRAME OR TABLE\n\n================\n\n")**

**stop(tempo.cat)**

**}**

**if( ! any(class(data2) %in% c("matrix", "data.frame", "table"))){**

**tempo.cat <- paste0("\n\n================\n\nERROR IN ", function.name, ": THE data2 ARGUMENT MUST BE A MATRIX, DATA FRAME OR TABLE\n\n================\n\n")**

**stop(tempo.cat)**

**}**

*# 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)) # activate this line and use the function to check arguments status*

*# end argument checking*

*# main code*

**same.class <- NULL**

**class <- NULL**

**same.dim <- NULL**

**dim <- NULL**

**same.row.nb <- NULL**

**row.nb <- NULL**

**same.col.nb <- NULL**

**col.nb <- NULL**

**same.row.name <- NULL**

**row.name <- NULL**

**any.id.row.name <- NULL**

**same.row.name.pos1 <- NULL**

**same.row.name.pos2 <- NULL**

**common.row.names <- NULL**

**same.col.name <- NULL**

**any.id.col.name <- NULL**

**same.col.name.pos1 <- NULL**

**same.col.name.pos2 <- NULL**

**common.col.names <- NULL**

**col.name <- NULL**

**any.id.row <- NULL**

**same.row.pos1 <- NULL**

**same.row.pos2 <- NULL**

**any.id.col <- NULL**

**same.col.pos1 <- NULL**

**same.col.pos2 <- NULL**

**identical.object <- NULL**

**identical.content <- NULL**

**if(identical(data1, data2) & any(class(data1) %in% c("matrix", "data.frame", "table"))){**

**same.class <- TRUE**

**class <- class(data1)**

**same.dim <- TRUE**

**dim <- dim(data1)**

**same.row.nb <- TRUE**

**row.nb <- nrow(data1)**

**same.col.nb <- TRUE**

**col.nb <- ncol(data1)**

**same.row.name <- TRUE**

**row.name <- dimnames(data1)[[1]]**

**any.id.row.name <- TRUE**

**same.row.name.pos1 <- 1:row.nb**

**same.row.name.pos2 <- 1:row.nb**

**common.row.names <- dimnames(data1)[[1]]**

**same.col.name <- TRUE**

**col.name <- dimnames(data1)[[2]]**

**any.id.col.name <- TRUE**

**same.col.name.pos1 <- 1:col.nb**

**same.col.name.pos2 <- 1:col.nb**

**common.col.names <- dimnames(data1)[[2]]**

**any.id.row <- TRUE**

**same.row.pos1 <- 1:row.nb**

**same.row.pos2 <- 1:row.nb**

**any.id.col <- TRUE**

**same.col.pos1 <- 1:col.nb**

**same.col.pos2 <- 1:col.nb**

**identical.object <- TRUE**

**identical.content <- TRUE**

**}else{**

**identical.object <- FALSE**

**if(all(class(data1) == "table") & length(dim(data1)) == 1){**

**tempo.cat <- paste0("\n\n================\n\nERROR IN ", function.name, ": THE data1 ARGUMENT IS A 1D TABLE. USE THE info\_1D\_dataset\_fun FUNCTION\n\n================\n\n")**

**stop(tempo.cat)**

**}**

**if(all(class(data2) == "table") & length(dim(data2)) == 1){**

**tempo.cat <- paste0("\n\n================\n\nERROR IN ", function.name, ": THE data2 ARGUMENT IS A 1D TABLE. USE THE info\_1D\_dataset\_fun FUNCTION\n\n================\n\n")**

**stop(tempo.cat)**

**}**

**if( ! identical(class(data1), class(data2))){**

**same.class <- FALSE**

**}else if( ! any(class(data1) %in% c("matrix", "data.frame", "table"))){**

**tempo.cat <- paste0("\n\n================\n\nERROR IN ", function.name, ": THE data1 AND data2 ARGUMENTS MUST BE EITHER MATRIX, DATA FRAME OR TABLE\n\n================\n\n")**

**stop(tempo.cat)**

**}else{**

**same.class <- TRUE**

**class <- class(data1)**

**}**

**if( ! identical(dim(data1), dim(data2))){**

**same.dim <- FALSE**

**}else{**

**same.dim <- TRUE**

**dim <- dim(data1)**

**}**

**if( ! identical(nrow(data1), nrow(data2))){**

**same.row.nb <- FALSE**

**}else{**

**same.row.nb <- TRUE**

**row.nb <- nrow(data1)**

**}**

**if( ! identical(ncol(data1), ncol(data2))){**

**same.col.nb <- FALSE**

**}else{**

**same.col.nb <- TRUE**

**col.nb <- ncol(data1)**

**}**

*# row and col names*

**if(is.null(dimnames(data1)) & is.null(dimnames(data2))){**

**same.row.name <- NULL**

**same.col.name <- NULL**

*# row and col names remain NULL*

**}else if((is.null(dimnames(data1)) & ! is.null(dimnames(data2))) | ( ! is.null(dimnames(data1)) & is.null(dimnames(data2)))){**

**same.row.name <- FALSE**

**same.col.name <- FALSE**

*# row and col names remain NULL*

**}else{**

**if( ! identical(dimnames(data1)[[1]], dimnames(data2)[[1]])){**

**same.row.name <- FALSE**

*# row names remain NULL*

**}else{**

**same.row.name <- TRUE**

**row.name <- dimnames(data1)[[1]]**

**}**

*# row names*

**any.id.row.name <- FALSE**

**if(any(dimnames(data1)[[1]] %in% dimnames(data2)[[1]])){**

**any.id.row.name <- TRUE**

**same.row.name.pos1 <- which(dimnames(data1)[[1]] %in% dimnames(data2)[[1]])**

**}**

**if(any(dimnames(data2)[[1]] %in% dimnames(data1)[[1]])){**

**any.id.row.name <- TRUE**

**same.row.name.pos2 <- which(dimnames(data2)[[1]] %in% dimnames(data1)[[1]])**

**}**

**if(any.id.row.name == TRUE){**

**common.row.names <- unique(c(dimnames(data1)[[1]][same.row.name.pos1], dimnames(data2)[[1]][same.row.name.pos2]))**

**}**

*# col names*

**any.id.col.name <- FALSE**

**if(any(dimnames(data1)[[2]] %in% dimnames(data2)[[2]])){**

**any.id.col.name <- TRUE**

**same.col.name.pos1 <- which(dimnames(data1)[[2]] %in% dimnames(data2)[[2]])**

**}**

**if(any(dimnames(data2)[[2]] %in% dimnames(data1)[[2]])){**

**any.id.col.name <- TRUE**

**same.col.name.pos2 <- which(dimnames(data2)[[2]] %in% dimnames(data1)[[2]])**

**}**

**if(any.id.col.name == TRUE){**

**common.col.names <- unique(c(dimnames(data1)[[2]][same.col.name.pos1], dimnames(data2)[[2]][same.col.name.pos2]))**

**}**

**if( ! identical(dimnames(data1)[[2]], dimnames(data2)[[2]])){**

**same.col.name <- FALSE**

*# col names remain NULL*

**}else{**

**same.col.name <- TRUE**

**col.name <- dimnames(data1)[[2]]**

**}**

**}**

*# identical row and col content*

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

**as.data.frame(matrix(data1, ncol = ncol(data1)), stringsAsFactors = FALSE)**

**}else if(all(class(data1) == "matrix")){**

**data1 <- as.data.frame(data1, stringsAsFactors = FALSE)**

**}else if(all(class(data1) == "data.frame")){**

**data1 <- data.frame(lapply(data1, as.character), stringsAsFactors=FALSE)**

**}**

**if(all(class(data2) == "table")){**

**as.data.frame(matrix(data2, ncol = ncol(data2)), stringsAsFactors = FALSE)**

**}else if(all(class(data2) == "matrix")){**

**data2 <- as.data.frame(data2, stringsAsFactors = FALSE)**

**}else if(all(class(data2) == "data.frame")){**

**data2 <- data.frame(lapply(data2, as.character), stringsAsFactors=FALSE)**

**}**

**row.names(data1) <- paste0("A", 1:nrow(data1))**

**row.names(data2) <- paste0("A", 1:nrow(data2))**

**if(same.col.nb == TRUE){** *# because if not the same col nb, the row cannot be identical*

**same.row.pos1 <- which(c(as.data.frame(t(data1), stringsAsFactors = FALSE)) %in% c(as.data.frame(t(data2), stringsAsFactors = FALSE)))**

**same.row.pos2 <- which(c(as.data.frame(t(data2), stringsAsFactors = FALSE)) %in% c(as.data.frame(t(data1), stringsAsFactors = FALSE)))**

**names(same.row.pos1) <- NULL**

**names(same.row.pos2) <- NULL**

**if(all(is.na(same.row.pos1))){**

**same.row.pos1 <- NULL**

**}else{**

**same.row.pos1 <- same.row.pos1[ ! is.na(same.row.pos1)]**

**any.id.row <- TRUE**

**}**

**if(all(is.na(same.row.pos2))){**

**same.row.pos2 <- NULL**

**}else{**

**same.row.pos2 <- same.row.pos2[ ! is.na(same.row.pos2)]**

**any.id.row <- TRUE**

**}**

**if(is.null(same.row.pos1) & is.null(same.row.pos2)){**

**any.id.row <- FALSE**

**}**

**}else{**

**any.id.row <- FALSE**

*# same.row.pos1 and 2 remain NULL*

**}**

**if(same.row.nb == TRUE){** *# because if not the same row nb, the col cannot be identical*

**same.col.pos1 <- which(c(data1) %in% c(data2))**

**same.col.pos2 <- which(c(data2) %in% c(data1))**

**names(same.col.pos1) <- NULL**

**names(same.col.pos2) <- NULL**

**if(all(is.na(same.col.pos1))){**

**same.col.pos1 <- NULL**

**}else{**

**same.col.pos1 <- same.col.pos1[ ! is.na(same.col.pos1)]**

**any.id.col <- TRUE**

**}**

**if(all(is.na(same.col.pos2))){**

**same.col.pos2 <- NULL**

**}else{**

**same.col.pos2 <- same.col.pos2[ ! is.na(same.col.pos2)]**

**any.id.col <- TRUE**

**}**

**if(is.null(same.col.pos1) & is.null(same.col.pos2)){**

**any.id.col <- FALSE**

**}**

**}else{**

**any.id.col <- FALSE**

*# same.col.pos1 and 2 remain NULL*

**}**

**if(same.dim == TRUE & ! all(is.null(same.row.pos1), is.null(same.row.pos2), is.null(same.col.pos1), is.null(same.col.pos2))){** *# same.dim == TRUE means that same.row.nb == TRUE and same.col.nb == TRUE, meaning that row.nb != NULL and col.nb != NULL. Thus, no need to include these checkings*

**if(identical(same.row.pos1, 1:row.nb) & identical(same.row.pos2, 1:row.nb) & identical(same.col.pos1, 1:col.nb) & identical(same.col.pos2, 1:col.nb)){**

**identical.content <- TRUE**

**}else{**

**identical.content <- FALSE**

**}**

**}else{**

**identical.content <- FALSE**

**}**

**}**

**output <- list(same.class = same.class, class = class, same.dim = same.dim, dim = dim, same.row.nb = same.row.nb, row.nb = row.nb, same.col.nb = same.col.nb , col.nb = col.nb, same.row.name = same.row.name, row.name = row.name, any.id.row.name = any.id.row.name, same.row.name.pos1 = same.row.name.pos1, same.row.name.pos2 = same.row.name.pos2, common.row.names = common.row.names, same.col.name = same.col.name, col.name = col.name,any.id.col.name = any.id.col.name, same.col.name.pos1 = same.col.name.pos1, same.col.name.pos2 = same.col.name.pos2, common.col.names = common.col.names, any.id.row = any.id.row, same.row.pos1 = same.row.pos1, same.row.pos2 = same.row.pos2, any.id.col = any.id.col, same.col.pos1 = same.col.pos1, same.col.pos2 = same.col.pos2, identical.object = identical.object, identical.content = identical.content)**

**return(output)**

**}**

######## fun\_2D\_head() #### head of the left or right of big 2D objects

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

**fun\_2D\_head <- function(data1, n = 10, side = "l"){**

*# AIM*

*# display the head of the left or right of big 2D objects*

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

*# fun\_param\_check()*

*# ARGUMENTS*

*# data1: matrix, data frame or table*

*# n: number of dimension to print (10 means 10 rows and columns)*

*# side: either "l" or "r" for the left or right side of the 2D object*

*# RETURN*

*# the head*

*# EXAMPLES*

*# obs1 = matrix(1:30, ncol = 5, dimnames = list(letters[1:6], LETTERS[1:5])) ; obs1 ; fun\_2D\_head(obs1, 3)*

*# obs1 = matrix(1:30, ncol = 5, dimnames = list(letters[1:6], LETTERS[1:5])) ; obs1 ; fun\_2D\_head(obs1, 3, "right")*

*# DEBUGGING*

*# data1 = matrix(1:30, ncol = 5) # for function debugging*

*# data1 = matrix(1:30, ncol = 5, dimnames = list(letters[1:2], LETTERS[1:5])) # for function debugging*

*# function name*

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

*# end function name*

*# required function checking*

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

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

**stop(tempo.cat)**

**}**

*# end required function checking*

*# argument checking*

*# argument checking without fun\_param\_check()*

**if( ! any(class(data1) %in% c("matrix", "data.frame", "table"))){**

**tempo.cat <- paste0("\n\n================\n\nERROR IN ", function.name, ": THE data1 ARGUMENT MUST BE A MATRIX, DATA FRAME OR TABLE\n\n================\n\n")**

**stop(tempo.cat)**

**}**

*# end argument checking without fun\_param\_check()*

*# argument checking with fun\_param\_check()*

**arg.check <- NULL** *# for function debbuging*

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

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

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

**tempo <- fun\_param\_check(data = side, options = c("l", "r"), length = 1, fun.name = function.name) ; eval(ee)**

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

**stop()** *# nothing else because print = TRUE by default in fun\_param\_check()*

**}**

*# end argument checking with fun\_param\_check()*

*# 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\_param\_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\_param\_check()*

*# end argument checking*

*# main code*

**obs.dim <- dim(data1)**

**row <- 1:ifelse(obs.dim[1] < n, obs.dim[1], n)**

**if(side == "l"){**

**col <- 1:ifelse(obs.dim[2] < n, obs.dim[2], n)**

**}**

**if(side == "r"){**

**col <- ifelse(obs.dim[2] < n, 1, obs.dim[2] - n + 1):obs.dim[2]**

**}**

**return(data1[row, col])**

**}**

######## fun\_2D\_tail() #### tail of the left or right of big 2D objects

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

**fun\_2D\_tail <- function(data1, n = 10, side = "l"){**

*# AIM*

*# display the tail of the left or right of big 2D objects*

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

*# fun\_param\_check()*

*# ARGUMENTS*

*# data1: matrix, data frame or table*

*# n: number of dimension to print (10 means 10 rows and columns)*

*# side: either "l" or "r" for the left or right side of the 2D object*

*# RETURN*

*# the tail*

*# EXAMPLES*

*# obs1 = matrix(1:30, ncol = 5, dimnames = list(letters[1:6], LETTERS[1:5])) ; obs1 ; fun\_2D\_tail(obs1, 3)*

*# obs1 = matrix(1:30, ncol = 5, dimnames = list(letters[1:6], LETTERS[1:5])) ; obs1 ; fun\_2D\_tail(obs1, 3, "r")*

*# DEBUGGING*

*# data1 = matrix(1:10, ncol = 5) # for function debugging*

*# data1 = matrix(1:10, ncol = 5, dimnames = list(letters[1:2], LETTERS[1:5])) # for function debugging*

*# function name*

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

*# end function name*

*# required function checking*

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

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

**stop(tempo.cat)**

**}**

*# end required function checking*

*# argument checking*

*# argument checking without fun\_param\_check()*

**if( ! any(class(data1) %in% c("matrix", "data.frame", "table"))){**

**tempo.cat <- paste0("\n\n================\n\nERROR IN ", function.name, ": THE data1 ARGUMENT MUST BE A MATRIX, DATA FRAME OR TABLE\n\n================\n\n")**

**stop(tempo.cat)**

**}**

*# end argument checking without fun\_param\_check()*

*# argument checking with fun\_param\_check()*

**arg.check <- NULL** *# for function debbuging*

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

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

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

**tempo <- fun\_param\_check(data = side, options = c("l", "r"), length = 1, fun.name = function.name) ; eval(ee)**

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

**stop()** *# nothing else because print = TRUE by default in fun\_param\_check()*

**}**

*# end argument checking with fun\_param\_check()*

*# 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\_param\_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\_param\_check()*

*# end argument checking*

*# main code*

**obs.dim <- dim(data1)**

**row <- ifelse(obs.dim[1] < n, 1, obs.dim[1] - n + 1):obs.dim[1]**

**if(side == "l"){**

**col <- 1:ifelse(obs.dim[2] < n, obs.dim[2], n)**

**}**

**if(side == "r"){**

**col <- ifelse(obs.dim[2] < n, 1, obs.dim[2] - n + 1):obs.dim[2]**

**}**

**return(data1[row, col])**

**}**

######## fun\_list\_comp() #### comparison of two lists

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

**fun\_list\_comp <- function(data1, data2){**

*# AIM*

*# compare two lists. Check and report in a list if the 2 datasets have:*

*# same length*

*# common names*

*# common compartments*

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

*# none*

*# ARGUMENTS*

*# data1: list*

*# data2: list*

*# RETURN*

*# a list containing:*

*# $same.length: logical. Are number of elements identical?*

*# $length: number of elements in the 2 datasets (NULL otherwise)*

*# $same.name: logical. Are element names identical ?*

*# $name: name of elements of the 2 datasets if identical (NULL otherwise)*

*# $any.id.name: logical. Is there any element names identical ?*

*# $same.name.pos1: position, in data1, of the element names identical in data2*

*# $same.name.pos2: position, in data2, of the compartment names identical in data1*

*# $any.id.compartment: logical. is there any identical compartments ?*

*# $same.compartment.pos1: position, in data1, of the compartments identical in data2*

*# $same.compartment.pos2: position, in data2, of the compartments identical in data1*

*# $identical.object: logical. Are objects identical (kind of object, compartment names and content)?*

*# $identical.content: logical. Are content objects identical (identical compartments excluding compartment names)?*

*# EXAMPLES*

*# obs1 = list(a = 1:5, b = LETTERS[1:2], d = matrix(1:6)) ; obs2 = list(a = 1:5, b = LETTERS[1:2], d = matrix(1:6)) ; fun\_list\_comp(obs1, obs2)*

*# obs1 = list(1:5, LETTERS[1:2]) ; obs2 = list(a = 1:5, b = LETTERS[1:2]) ; fun\_list\_comp(obs1, obs2)*

*# obs1 = list(b = 1:5, c = LETTERS[1:2]) ; obs2 = list(a = 1:5, b = LETTERS[1:2], d = matrix(1:6)) ; fun\_list\_comp(obs1, obs2)*

*# obs1 = list(b = 1:5, c = LETTERS[1:2]) ; obs2 = list(LETTERS[5:9], matrix(1:6), 1:5) ; fun\_list\_comp(obs1, obs2)*

*# DEBUGGING*

*# data1 = list(a = 1:5, b = LETTERS[1:2], d = matrix(1:6)) ; data2 = list(a = 1:5, b = LETTERS[1:2], d = matrix(1:6)) # for function debugging*

*# data1 = list(a = 1:5, b = LETTERS[1:2]) ; data2 = list(a = 1:5, b = LETTERS[1:2], d = matrix(1:6)) # for function debugging*

*# function name*

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

*# end function name*

*# argument checking*

**if( ! any(class(data1) %in% "list")){**

**tempo.cat <- paste0("\n\n================\n\nERROR IN ", function.name, ": THE data1 ARGUMENT MUST BE A LIST\n\n================\n\n")**

**stop(tempo.cat)**

**}**

**if( ! any(class(data2) %in% "list")){**

**tempo.cat <- paste0("\n\n================\n\nERROR IN ", function.name, ": THE data2 ARGUMENT MUST BE A LIST\n\n================\n\n")**

**stop(tempo.cat)**

**}**

*# 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)) # activate this line and use the function to check arguments status*

*# end argument checking*

*# main code*

**same.length <- NULL**

**length <- NULL**

**same.name <- NULL**

**name <- NULL**

**any.id.name <- NULL**

**same.name.pos1 <- NULL**

**same.name.pos2 <- NULL**

**any.id.compartment <- NULL**

**same.compartment.pos1 <- NULL**

**same.compartment.pos2 <- NULL**

**identical.object <- NULL**

**identical.content <- NULL**

**if(identical(data1, data2)){**

**same.length <- TRUE**

**length <- length(data1)**

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

**same.name <- TRUE**

**name <- names(data1)**

**any.id.name <- TRUE**

**same.name.pos1 <- 1:length(data1)**

**same.name.pos2 <- 1:length(data2)**

**}**

**any.id.compartment <- TRUE**

**same.compartment.pos1 <- 1:length(data1)**

**same.compartment.pos2 <- 1:length(data2)**

**identical.object <- TRUE**

**identical.content <- TRUE**

**}else{**

**identical.object <- FALSE**

**if( ! identical(length(data1), length(data2))){**

**same.length<- FALSE**

**}else{**

**same.length<- TRUE**

**length <- length(data1)**

**}**

**if( ! (is.null(names(data1)) & is.null(names(data2)))){**

**if( ! identical(names(data1), names(data2))){**

**same.name <- FALSE**

**}else{**

**same.name <- TRUE**

**name <- names(data1)**

**}**

**any.id.name <- FALSE**

**if(any(names(data1) %in% names(data2))){**

**any.id.name <- TRUE**

**same.name.pos1 <- which(names(data1) %in% names(data2))**

**}**

**if(any(names(data2) %in% names(data1))){**

**any.id.name <- TRUE**

**same.name.pos2 <- which(names(data2) %in% names(data1))**

**}**

**}**

**names(data1) <- NULL**

**names(data2) <- NULL**

**any.id.compartment <- FALSE**

**if(any(data1 %in% data2)){**

**any.id.compartment <- TRUE**

**same.compartment.pos1 <- which(data1 %in% data2)**

**}**

**if(any(data2 %in% data1)){**

**any.id.compartment <- TRUE**

**same.compartment.pos2 <- which(data2 %in% data1)**

**}**

**if(same.length == TRUE & ! all(is.null(same.compartment.pos1), is.null(same.compartment.pos2))){**

**if(identical(same.compartment.pos1, same.compartment.pos2)){**

**identical.content <- TRUE**

**}else{**

**identical.content <- FALSE**

**}**

**}else{**

**identical.content <- FALSE**

**}**

**}**

**output <- list(same.length = same.length, length = length, same.name = same.name, name = name, any.id.name = any.id.name, same.name.pos1 = same.name.pos1, same.name.pos2 = same.name.pos2, any.id.compartment = any.id.compartment, same.compartment.pos1 = same.compartment.pos1, same.compartment.pos2 = same.compartment.pos2, identical.object = identical.object, identical.content = identical.content)**

**return(output)**

**}**

################ Object modification

######## fun\_dataframe\_remodeling() #### remodeling a data frame to have column name as a qualitative column and vice-versa

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

**fun\_dataframe\_remodeling <- function(data, quanti.col.name = "quanti", quali.col.name = "quali"){**

*# AIM*

*# if the data frame is made of numeric columns, a new data frame is created, with the 1st column gathering all the numeric values, and the 2nd column being the name of the columns of the initial data frame. If row names were present in the initial data frame, then a new ini\_rowname column is added with the names of the rows*



*# If the data frame is made of one numeric column and one character or factor column, a new data frame is created, with the new columns corresponding to the split numeric values (according to the character column). NA are added a the end of each column to have the same number of rows. BEWARE: in such data frame, rows are not individuals. This means that in the example below, values 10 and 20 are associated on the same row but that means nothing in term of association*



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

*# fun\_param\_check()*

*# ARGUMENTS*

*# data: data frame to convert*

*# quanti.col.name: optional name for the quanti column of the new data frame*

*# quali.col.name: optional name for the quali column of the new data frame*

*# RETURN*

*# the modified data frame*

*# EXAMPLES*

*# obs <- data.frame(col1 = (1:4)\*10, col2 = c("A", "B", "A", "A")) ; obs ; fun\_dataframe\_remodeling(obs)*

*# obs <- data.frame(col1 = (1:4)\*10, col2 = 5:8) ; obs ; fun\_dataframe\_remodeling(obs, quanti.col.name = "quanti", quali.col.name = "quali")*

*# obs <- data.frame(col1 = (1:4)\*10, col2 = 5:8) ; rownames(obs) <- paste0("row", 1:4) ; obs ; fun\_dataframe\_remodeling(obs, quanti.col.name = "quanti", quali.col.name = "quali")*

*# DEBUGGING*

*# data = data.frame(a = 1:3, b = 4:6) ; quanti.col.name = "quanti" ; quali.col.name = "quali" # for function debugging*

*# data = data.frame(a = 1:3, b = 4:6, c = 11:13) ; quanti.col.name = "quanti" ; quali.col.name = "quali" # for function debugging*

*# data = data.frame(a = 1:3, b = letters[1:3]) ; quanti.col.name = "quanti" ; quali.col.name = "quali" # for function debugging*

*# data = data.frame(a = 1:3, b = letters[1:3]) ; quanti.col.name = "TEST" ; quali.col.name = "quali" # for function debugging*

*# data = data.frame(b = letters[1:3], a = 1:3) ; quanti.col.name = "quanti" ; quali.col.name = "quali" # for function debugging*

*# data = data.frame(b = c("e", "e", "h"), a = 1:3) ; quanti.col.name = "quanti" ; quali.col.name = "quali" # for function debugging*

*# function name*

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

*# end function name*

*# required function checking*

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

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

**stop(tempo.cat)**

**}**

*# end required function checking*

*# argument checking*

*# argument checking without fun\_param\_check()*

**if( ! any(class(data) %in% "data.frame")){**

**tempo.cat <- paste0("\n\n================\n\nERROR IN ", function.name, ": THE data ARGUMENT MUST BE A DATA FRAME\n\n================\n\n")**

**stop(tempo.cat)**

**}**

*# end argument checking without fun\_param\_check()*

*# argument checking with fun\_param\_check()*

**arg.check <- NULL** *# for function debbuging*

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

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

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

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

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

**stop()** *# nothing else because print = TRUE by default in fun\_param\_check()*

**}**

*# end argument checking with fun\_param\_check()*

*# 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\_param\_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\_param\_check()*

*# end argument checking*

*# main code*

**tempo.factor <- unlist(lapply(data, class))**

**for(i in 1:length(tempo.factor)){** *# convert factor columns as character*

**if(all(tempo.factor[i] == "factor")){**

**data[, i] <- as.character(data[, i])**

**}**

**}**

**tempo.factor <- unlist(lapply(data, mode))**

**if(length(data) == 2){**

**if( ! ((mode(data[, 1]) == "character" & mode(data[, 2]) == "numeric") | mode(data[, 2]) == "character" & mode(data[, 1]) == "numeric" | mode(data[, 2]) == "numeric" & mode(data[, 1]) == "numeric") ){**

**tempo.cat <- paste0("\n\n================\n\nERROR IN ", function.name, ": IF data ARGUMENT IS A DATA FRAME MADE OF 2 COLUMNS, EITHER A COLUMN MUST BE NUMERIC AND THE OTHER CHARACTER, OR THE TWO COLUMNS MUST BE NUMERIC\n\n================\n\n")**

**stop(tempo.cat)**

**}**

**if((mode(data[, 1]) == "character" | mode(data[, 2]) == "character") & (quanti.col.name != "quanti" | quali.col.name != "quali")){**

**tempo.cat <- paste0("\n\n================\n\nERROR IN ", function.name, ": IMPROPER quanti.col.name OR quali.col.name RESETTINGS. THESE ARGUMENTS ARE RESERVED FOR DATA FRAMES MADE OF n NUMERIC COLUMNS ONLY\n\n================\n\n")**

**stop(tempo.cat)**

**}**

**}else{**

**if( ! all(tempo.factor %in% "numeric")){**

**tempo.cat <- paste0("\n\n================\n\nERROR IN ", function.name, ": IF data ARGUMENT IS A DATA FRAME MADE OF ONE COLUMN, OR MORE THAN 2 COLUMNS, THESE COLUMNS MUST BE NUMERIC\n\n================\n\n")**

**stop(tempo.cat)**

**}**

**}**

**if(( ! any(tempo.factor %in% "character")) & is.null(names(data))){**

**tempo.cat <- paste0("\n\n================\n\nERROR IN ", function.name, ": NUMERIC DATA FRAME in the data ARGUMENT MUST HAVE COLUMN NAMES\n\n================\n\n")**

**stop()**

**}**

**if(all(tempo.factor %in% "numeric")){** *# transfo 1*

**quanti <- NULL**

**for(i in 1:length(data)){**

**quanti <-c(quanti, data[, i])**

**}**

**quali <- rep(names(data), each = nrow(data))**

**output.data <- data.frame(quanti, quali)**

**names(output.data) <- c(quanti.col.name, quali.col.name)**

*# add the ini\_rowname column*

**ini.rownames <- rownames(data)**

**tempo.data <- data**

**rownames(tempo.data) <- NULL**

**null.rownames <- (tempo.data)**

**if( ! identical(ini.rownames, null.rownames)){**

**ini\_rowname <- rep(ini.rownames, times = ncol(data))**

**output.data <- cbind(output.data, ini\_rowname)**

**}**

**}else{** *# transfo 2*

**if(class(data[, 1]) == "character"){**

**data <- cbind(data[2], data[1])**

**}**

**nc.max <- max(table(data[, 2]))** *# effectif maximum des classes*

**nb.na <- nc.max - table(data[,2])** *# nombre de NA à ajouter pour réaliser la data frame*

**tempo<-split(data[, 1], data[, 2])**

**for(i in 1:length(tempo)){tempo[[i]] <- append(tempo[[i]], rep(NA, nb.na[i]))}** *# des NA doivent être ajoutés lorsque les effectifs sont différents entre les classes. C'est uniquement pour que chaque colonne ait le même nombre de lignes*

**output.data<-data.frame(tempo)**

**}**

**return(output.data)**

**}**

######## fun\_refactorization() #### remove classes that are not anymore present in factors or factor columns in data frames

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

**fun\_refactorization <- function(data, also.ordered = TRUE){**

*# AIM*

*# refactorize a factor or the factor columns of a data frame, such as only the class present are in the levels (no empty levels). The class order in levels is kept. Do not work on character vector or column of data frame*

*# useful to remove the empty classes after row removing for instance*

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

*# fun\_param\_check()*

*# ARGUMENTS*

*# data: factor (ordered or not) or data frame*

*# also.ordered: refactorize also ordered factors? This to deal with ordered factors that have class "ordered" "factor"*

*# RETURN*

*# a list containing:*

*# $data: the modified object*

*# $removed: the removed classes for a factor and a list of the removed classes for each factor class of the data frame*

*# EXAMPLES*

*# obs <- data.frame(a = LETTERS[1:6], b = paste0(letters[1.6], c(1,1,2,2,3,3)), c = ordered(LETTERS[7:12]), d = 1:6, e = "A")[-c(1:2),] ; sapply(obs, levels) ; fun\_refactorization(obs, FALSE)*

*# obs <- data.frame(a = LETTERS[1:6], b = paste0(letters[1.6], c(1,1,2,2,3,3)), c = ordered(LETTERS[7:12]), d = 1:6, e = "A")[-c(1:2),] ; sapply(obs, levels) ; fun\_refactorization(obs, TRUE)*

*# obs <- factor(LETTERS[1:6])[-c(1:2)] ; obs ; fun\_refactorization(obs, TRUE)*

*# obs <- ordered(LETTERS[1:6])[-c(1:2)] ; obs ; fun\_refactorization(obs, TRUE)*

*# obs <- factor(LETTERS[1:6], levels = rev(LETTERS[1:6]))[-c(1:2)] ; obs ; fun\_refactorization(obs, FALSE)*

*# DEBUGGING*

*# data <- data.frame(a = LETTERS[1:6], b = paste0(letters[1.6], c(1,1,2,2,3,3)), c = ordered(LETTERS[7:12]), d = 1:6, e = "A") ; data <- data[-c(1:2),] ; also.ordered <- TRUE # for function debugging*

*# data <- factor(LETTERS[1:6])[-c(1:2)] ; also.ordered <- TRUE # for function debugging*

*# data <- ordered(LETTERS[1:6])[-c(1:2)] ; also.ordered <- TRUE # for function debugging*

*# function name*

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

*# end function name*

*# required function checking*

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

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

**stop(tempo.cat)**

**}**

*# end required function checking*

*# argument checking*

*# argument checking with fun\_param\_check()*

**arg.check <- NULL** *# for function debbuging*

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

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

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

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

**stop()** *# nothing else because print = TRUE by default in fun\_param\_check()*

**}**

*# end argument checking with fun\_param\_check()*

*# argument checking without fun\_param\_check()*

**if(also.ordered == FALSE){**

**if( ! (all(class(data) == "data.frame") | all(class(data) == "factor"))){**

**tempo.cat <- paste0("\n\n================\n\nERROR IN ", function.name, ": data ARGUMENT MUST BE A FACTOR (NON ORDERED BECAUSE THE also.ordered ARGUMENT IS SET TO FALSE) OR A DATA FRAME\n\n================\n\n")**

**stop(tempo.cat)**

**}**

**}**

**if(also.ordered == TRUE){**

**if( ! (all(class(data) == "data.frame") | all(class(data) == "factor") | all(class(data) %in% c("ordered", "factor")))){**

**tempo.cat <- paste0("\n\n================\n\nERROR IN ", function.name, ": data ARGUMENT MUST BE A FACTOR OR A DATA FRAME\n\n================\n\n")**

**stop(tempo.cat)**

**}**

**}**

*# end argument checking without fun\_param\_check()*

*# 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\_param\_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\_param\_check()*

*# end argument checking*

*# main code*

**text <- NULL**

**if(all(class(data) == "factor")){**

**tempo.keep.log <- levels(data) %in% unique(data)**

**text <- levels(data)[ ! tempo.keep.log]**

**data <- factor(data, levels = levels(data)[tempo.keep.log])**

**}else if(all(class(data) %in% c("ordered", "factor"))){**

**tempo.keep.log <- levels(data) %in% unique(data)**

**text <- levels(data)[ ! tempo.keep.log]**

**data <- ordered(data, levels = levels(data)[tempo.keep.log])**

**}else if(all(class(data) == "data.frame")){**

**text <- vector("list", length(data))**

**names(text) <- names(data)**

**tempo.factor.col <- sapply(sapply(lapply(data, class), FUN = "%in%", "factor"), FUN = "all")** *# get the factor column (logical)*

**for(i in 1:length(tempo.factor.col)){**

**if(tempo.factor.col[i] == TRUE){**

**tempo.keep.log <- levels(data[[i]]) %in% unique(data[[i]])**

**text[[i]] <- levels(data[[i]])[ ! tempo.keep.log]**

**data[[i]] <- factor(data[[i]], levels = levels(data[[i]])[tempo.keep.log])**

**}**

**}**

**tempo.ordered.col <- sapply(sapply(lapply(data, class), FUN = "%in%", "ordered"), FUN = "any")** *# get the ordered factor column (logical) if they exist*

**if(also.ordered == TRUE){**

**for(i in 1:length(tempo.ordered.col)){**

**if(tempo.ordered.col[i] == TRUE){**

**tempo.keep.log <- levels(data[[i]]) %in% unique(data[[i]])**

**text[[i]] <- levels(data[[i]])[ ! tempo.keep.log]**

**data[[i]] <- ordered(data[[i]], levels = levels(data[[i]])[tempo.keep.log])**

**}**

**}**

**}**

**text <- text[(tempo.factor.col | tempo.ordered.col) & ! (sapply(text, FUN = length) == 0)]** *# remove the compartments of text that are not modified factors columns of data frame*

**}**

**output <- list(data = data, removed = text)**

**return(output)**

**}**

######## fun\_round() #### Rounding number if decimal present

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

**fun\_round <- function(data, dec.nb = 2, after.lead.zero = TRUE){**

*# AIM*

*# round a vector of values, if decimal, with the desired number of decimal digits after the decimal leading zeros*

*# BEWARE*

*# Work well with numbers as character strings, but not always with numerical numbers because of the floating point*

*# Numeric values are really truncated from a part of their decimal digits, whatever options(digits) settings*

*# ARGUMENTS*

*# data: a vector of numbers (numeric or character mode)*

*# dec.nb: number of required decimal digits*

*# after.lead.zero: logical. If FALSE, rounding is performed for all the decimal numbers, whatever the leading zeros (e.g., 0.123 -> 0.12 and 0.00128 -> 0.00). If TRUE, dec.nb are taken after the leading zeros (e.g., 0.123 -> 0.12 and 0.00128 -> 0.0013)*

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

*# fun\_param\_check()*

*# RETURN*

*# the modified vector*

*# EXAMPLES*

*# cat(fun\_round(data = c(10, 100.001, 333.0001254, 12312.1235), dec.nb = 2, after.lead.zero = FALSE), "\n\n")*

*# cat(fun\_round(data = c("10", "100.001", "333.0001254", "12312.1235"), dec.nb = 2, after.lead.zero = FALSE), "\n\n")*

*# cat(fun\_round(data = c("10", "100.001", "333.0001254", "12312.1235"), dec.nb = 2, after.lead.zero = TRUE), "\n\n")*

*# DEBUGGING*

*# data = data = c(10, 100.001, 333.0001254, 12312.1235) ; dec.nb = 2 ; after.lead.zero = FALSE # # for function debugging*

*# data = data = c("10", "100.001", "333.0001254", "12312.1235") ; dec.nb = 2 ; after.lead.zero = TRUE # # for function debugging*

*# function name*

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

*# end function name*

*# required function checking*

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

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

**stop(tempo.cat)**

**}**

*# end required function checking*

*# argument checking*

*# argument checking without fun\_param\_check()*

**if( ! (all(typeof(data) == "character") | all(typeof(data) == "double") | all(typeof(data) == "integer"))){**

**tempo.cat <- paste0("\n\n================\n\nERROR IN ", function.name, ": data ARGUMENT MUST BE A VECTOR OF NUMBERS (IN NUMERIC OR CHARACTER MODE)\n\n================\n\n")**

**stop(tempo.cat)**

**}**

*# end argument checking without fun\_param\_check()*

*# argument checking with fun\_param\_check()*

**arg.check <- NULL** *# for function debbuging*

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

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

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

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

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

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

**stop()** *# nothing else because print = TRUE by default in fun\_param\_check()*

**}**

*# end argument checking with fun\_param\_check()*

*# 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\_param\_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\_param\_check()*

*# end argument checking*

*# main code*

**tempo <- grepl(x = data, pattern = "\\.")** *# detection of decimal numbers*

**ini.mode <- mode(data)**

**data <- as.character(data)** *# to really truncate decimal digits*

**for(i in 1:length(data)){** *# scan all the numbers of the vector*

**if(tempo[i] == TRUE){** *# means decimal number*

**if(after.lead.zero == TRUE){**

**zero.pos <- unlist(gregexpr(text=data[i], pattern = 0))** *# recover all the position of the zeros in the number. -1 if no zeros (do not record the leading and trailing zeros)*

**}else{**

**zero.pos <- -1** *# -1 as if no zero*

**}**

**dot.pos <- unlist(gregexpr(text=data[i], pattern = "\\."))** *# recover all the position of the zeros in the number*

**digit.pos <- unlist(gregexpr(text=data[i], pattern = "[[:digit:]]"))** *# recover all the position of the digits in the number*

**dec.pos <- digit.pos[digit.pos > dot.pos]**

**count <- 0**

**while((dot.pos + count + 1) %in% zero.pos & (dot.pos + count + 1) <= max(dec.pos) & (count + dec.nb) < length(dec.pos)){** *# count the number of leading zeros in the decimal part*

**count <- count + 1**

**}**

**data[i] <- formatC(as.numeric(data[i]), digits = (count + dec.nb), format = "f")**

**}**

**}**

**if(ini.mode != "character"){**

**data <- as.numeric(data)**

**}**

**return(data)**

**}**

######## fun\_90clock\_matrix\_rot() #### 90° clockwise matrix rotation

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

**fun\_90clock\_matrix\_rot <- function(data){**

*# AIM*

*# 90° clockwise matrix rotation*

*# applied twice, the function provide the mirror matrix, according to vertical and horizontal symmetry*

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

*# fun\_param\_check()*

*# ARGUMENTS*

*# data: matrix (matrix class)*

*# RETURN*

*# the modified matrix*

*# EXAMPLES*

*# obs <- matrix(1:10, ncol = 1) ; obs ; fun\_90clock\_matrix\_rot(obs)*

*# obs <- matrix(LETTERS[1:10], ncol = 5) ; obs ; fun\_90clock\_matrix\_rot(obs)*

*# DEBUGGING*

*# data = matrix(1:10, ncol = 1)*

*# function name*

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

*# end function name*

*# required function checking*

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

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

**stop(tempo.cat)**

**}**

*# end required function checking*

*# argument checking*

**arg.check <- NULL** *# for function debbuging*

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

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

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

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

**stop()** *# nothing else because print = TRUE by default in fun\_param\_check()*

**}**

*# 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\_param\_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\_param\_check()*

*# end argument checking*

*# main code*

**for (i in 1:ncol(data)){data[,i] <- rev(data[,i])}**

**data <- t(data)**

**return(data)**

**}**

######## fun\_num2color\_mat() #### Conversion of a numeric matrix into hexadecimal color matrix

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

**fun\_num2color\_mat <- function(mat1, mat.hsv.h = TRUE, notch = 1, s = 1, v = 1, forced.color = NULL){**

*# AIM*

*# convert a matrix made of numbers into a hexadecimal matrix for rgb colorization*

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

*# fun\_param\_check()*

*# ARGUMENTS:*

*# mat1: matrix 1 of non negative numerical values that has to be colored (matrix class). NA allowed*

*# mat.hsv.h: logical. Is mat1 the h of hsv colors ? (if TRUE, mat1 must be between zero and 1)*

*# notch: single value between 0 and 1 to shift the successive colors on the hsv circle by + notch*

*# s: s argument of hsv(). Must be between 0 and 1*

*# v: v argument of hsv(). Must be between 0 and 1*

*# forced.color: Must be NULL or hexadecimal color code or name given by colors(). The first minimal values of mat1 will be these colors. All the color of mat1 can be forced using this argument*

*# RETURN*

*# a list containing:*

*# $mat1.name: name of mat1*

*# $colored.mat: colors of mat1 in hexa*

*# $problem: logical. Is any colors of forced.color overlap the colors designed by the function. NULL if forced.color = NULL*

*# $text.problem: text when overlapping colors. NULL if forced.color = NULL or problem == FALSE*

*# EXAMPLES*

*# mat1 = matrix(c(1,1,1,2,1,5,9,NA), ncol = 2) ; dimnames(mat1) <- list(LETTERS[1:4], letters[1:2]) ; fun\_num2color\_mat(mat1, mat.hsv.h = FALSE, notch = 1, s = 1, v = 1, forced.color = NULL)*

*# DEBUGGING*

*# mat1 = matrix(c(1,1,1,2,1,5,9,NA), ncol = 2) ; dimnames(mat1) <- list(LETTERS[1:4], letters[1:2]); mat.hsv.h = FALSE ; notch = 1 ; s = 1 ; v = 1 ; forced.color = c(hsv(1,1,1), hsv(0,0,0)) # for function debugging*

*# function name*

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

*# end function name*

*# required function checking*

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

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

**stop(tempo.cat)**

**}**

*# end required function checking*

*# argument checking*

*# argument checking with fun\_param\_check()*

**arg.check <- NULL** *# for function debbuging*

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

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

**tempo <- fun\_param\_check(data = mat1, mode = "numeric", class = "matrix", na.contain = TRUE, neg.values = FALSE, fun.name = function.name) ; eval(ee)**

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

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

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

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

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

**stop()** *# nothing else because print = TRUE by default in fun\_param\_check()*

**}**

*# end argument checking with fun\_param\_check()*

*# argument checking without fun\_param\_check()*

**if(mat.hsv.h == TRUE & fun\_param\_check(data = mat1, mode = "numeric", prop = TRUE, print = FALSE)$problem == TRUE){**

**tempo.cat <- paste0("\n\n================\n\nERROR IN ", function.name, ": mat1 ARGUMENT MUST BE A MATRIX OF PROPORTIONS SINCE THE mat.hsv.h ARGUMENT IS SET TO TRUE\n\n================\n\n")**

**stop(tempo.cat)**

**}**

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

**tempo <- fun\_param\_check(data = forced.color, class = "character")**

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

**stop()**

**}**

**if( ! all(forced.color %in% colors() | grepl(pattern = "^#", forced.color))){** *# check that all strings of forced.color start by #*

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

**stop(tempo.cat)**

**}**

**}**

*# end argument checking without fun\_param\_check()*

*# 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\_param\_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\_param\_check()*

*# end argument checking*

*# main code*

**problem <- NULL**

**text.problem <- NULL**

**mat1.name <- deparse(substitute(mat1))**

*# change the scale of the plotted matrix*

**if(mat.hsv.h == TRUE){**

**if(any(min(mat1, na.rm = TRUE) < 0 | max(mat1, na.rm = TRUE) > 1, na.rm = TRUE)){**

**tempo.cat <- paste0("\n\n================\n\nERROR IN ", function.name, ": mat1 MUST BE MADE OF VALUES BETWEEN 0 AND 1 BECAUSE mat.hsv.h ARGUMENT SET TO TRUE\n\n================\n\n")**

**stop(tempo.cat)**

**}**

**}else{**

**if(any(mat1 - floor(mat1) > 0, na.rm = TRUE) | any(mat1 == 0, na.rm = TRUE)){**

**tempo.cat <- paste0("\n\n================\n\nERROR IN ", function.name, ": mat1 MUST BE MADE OF INTEGER VALUES WITHOUT 0 BECAUSE mat.hsv.h ARGUMENT SET TO FALSE\n\n================\n\n")**

**stop(tempo.cat)**

**}else{**

**mat1 <- mat1 / max(mat1, na.rm = TRUE)**

**}**

**}**

**if(notch != 1){**

**different.color <- unique(as.vector(mat1))**

**different.color <- different.color[ ! is.na(different.color)]**

**tempo.different.color <- different.color + c(0, cumsum(rep(notch, length(different.color) - 1)))**

**tempo.different.color <- tempo.different.color - floor(tempo.different.color)**

**if(any(duplicated(tempo.different.color) == TRUE)){**

**tempo.cat <- paste0("\n\n================\n\nERROR IN ", function.name, ": DUPLICATED VALUES AFTER USING notch (", paste(tempo.different.color[duplicated(tempo.different.color)], collapse = " "), "). TRY ANOTHER notch VALUE\n\n================\n\n")**

**stop(tempo.cat)**

**}else if(length(different.color) != length(tempo.different.color)){**

**tempo.cat <- paste0("\n\n================\n\nERROR IN ", function.name, ": LENGTH OF different.color (", paste(different.color, collapse = " "), ") DIFFERENT FROM LENGTH OF tempo.different.color (", paste(tempo.different.color, collapse = " "), ")\n\n================\n\n")**

**stop(tempo.cat)**

**}else{**

**for(i in 1:length(different.color)){**

**mat1[mat1 == different.color[i]] <- tempo.different.color[i]**

**}**

**}**

**}**

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

**hexa.values.to.change <- hsv(unique(sort(mat1))[1:length(forced.color)], s, v)**

**}**

**mat1[ ! is.na(mat1)] <- hsv(mat1[ ! is.na(mat1)], s, v)**

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

**if(any(forced.color %in% mat1, na.rm = TRUE)){**

**problem <- TRUE**

**text.problem <- paste0("THE FOLLOWING COLORS WHERE INTRODUCED USING forced.color BUT WHERE ALREADY PRESENT IN THE COLORED MATRIX :", paste(forced.color[forced.color %in% mat1], collapse = " "))**

**}else{**

**problem <- FALSE**

**}**

**for(i in 1:length(hexa.values.to.change)){**

**if( ! any(mat1 == hexa.values.to.change[i], na.rm = TRUE)){**

**tempo.cat <- paste0("\n\n================\n\nERROR IN ", function.name, ": THE ", hexa.values.to.change[i], " VALUE FROM hexa.values.to.change IS NOT REPRESENTED IN mat1 : ", paste(unique(as.vector(mat1)), collapse = " "), "\n\n================\n\n")**

**stop(tempo.cat)**

**}else{**

**mat1[which(mat1 == hexa.values.to.change[i])] <- forced.color[i]**

**}**

**}**

**}**

**output <- list(mat1.name = mat1.name, colored.mat = mat1, problem = problem, text.problem = text.problem)**

**return(output)**

**}**

######## fun\_by\_case\_matrix\_op() #### assembling of several matrices with operation

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

**fun\_by\_case\_matrix\_op <- function(mat.list, kind.of.operation = "+"){**

*# AIM*

*# assemble several matrices of same dimensions by performing by case operation. For instance add the value of all the case 1 (row1 & column1) of the matrices and put it in the case 1 of a new matrix M, add the value of all the case 2 (row2 & column1) of the matrices and put it in the case 2 of a new matrix M, etc.*



*# c: case*

*# i: row number*

*# j: column number*

*# k: matrix number*

*# z: number of matrices*

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

*# fun\_param\_check()*

*# fun\_2D\_comp()*

*# ARGUMENTS:*

*# mat.list: list of matrices*

*# kind.of.operation: either "+" (by case addition), "-" (by case subtraction) or "\*" (by case multiplication)*

*# RETURN*

*# the assembled matrix, with row and/or column names only if all the matrices have identical row/column names*

*# EXAMPLES*

*# mat1 = matrix(c(1,1,1,2,1,5,9,8), ncol = 2) ; mat2 = matrix(c(1,1,1,2,1,5,9,NA), ncol = 2) ; fun\_by\_case\_matrix\_op(mat.list = list(mat1, mat2), kind.of.operation = "+")*

*# mat1 = matrix(c(1,1,1,2,1,5,9,8), ncol = 2, dimnames = list(LETTERS[1:4], letters[1:2])) ; mat2 = matrix(c(1,1,1,2,1,5,9,NA), ncol = 2, dimnames = list(LETTERS[1:4], letters[1:2])) ; fun\_by\_case\_matrix\_op(mat.list = list(mat1, mat2), kind.of.operation = "\*")*

*# mat1 = matrix(c(1,1,1,2,1,5,9,8), ncol = 2, dimnames = list(LETTERS[1:4], c(NA, NA))) ; mat2 = matrix(c(1,1,1,2,1,5,9,NA), ncol = 2, dimnames = list(LETTERS[1:4], letters[1:2])) ; fun\_by\_case\_matrix\_op(mat.list = list(mat1, mat2), kind.of.operation = "-")*

*# mat1 = matrix(c(1,1,1,2,1,5,9,8), ncol = 2, dimnames = list(c("A1", "A2", "A3", "A4"), letters[1:2])) ; mat2 = matrix(c(1,1,1,2,1,5,9,NA), ncol = 2, dimnames = list(LETTERS[1:4], letters[1:2])) ; mat3 = matrix(c(1,1,1,2,1,5,9,NA), ncol = 2, dimnames = list(LETTERS[1:4], letters[1:2])) ; fun\_by\_case\_matrix\_op(mat.list = list(mat1, mat2, mat3), kind.of.operation = "+")*

*# DEBUGGING*

*# mat1 = matrix(c(1,1,1,2,1,5,9,8), ncol = 2) ; mat2 = matrix(c(1,1,1,2,1,5,9,NA), ncol = 2) ; mat.list = list(mat1, mat2) ; kind.of.operation = "+" # for function debugging*

*# mat1 = matrix(c(1,1,1,2,1,5,9,8), ncol = 2, dimnames = list(LETTERS[1:4], c(NA, NA))) ; mat2 = matrix(c(1,1,1,2,1,5,9,NA), ncol = 2, dimnames = list(LETTERS[1:4], letters[1:2])) ; mat.list = list(mat1, mat2) ; kind.of.operation = "\*" # for function debugging*

*# function name*

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

*# end function name*

*# required function checking*

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

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

**stop(tempo.cat)**

**}**

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

**tempo.cat <- paste0("\n\n================\n\nERROR IN ", function.name, ": REQUIRED fun\_2D\_comp() FUNCTION IS MISSING IN THE R ENVIRONMENT\n\n================\n\n")**

**stop(tempo.cat)**

**}**

*# end required function checking*

*# argument checking*

*# argument checking with fun\_param\_check()*

**arg.check <- NULL** *# for function debbuging*

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

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

**tempo <- fun\_param\_check(data = mat.list, class = "list", fun.name = function.name) ; eval(ee)**

**tempo <- fun\_param\_check(data = kind.of.operation, options = c("+", "-", "\*"), length = 1, fun.name = function.name) ; eval(ee)**

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

**stop()** *# nothing else because print = TRUE by default in fun\_param\_check()*

**}**

*# end argument checking with fun\_param\_check()*

*# argument checking without fun\_param\_check()*

**if(length(mat.list) < 2){**

**tempo.cat <- paste0("\n\n================\n\nERROR IN ", function.name, ": mat.list ARGUMENT MUST BE A LIST CONTAINING AT LEAST 2 MATRICES\n\n================\n\n")**

**stop(tempo.cat)**

**}**

**for(i0 in 1:length(mat.list)){**

**tempo <- fun\_param\_check(data = mat.list[[i0]], class = "matrix", mode = "numeric", na.contain = TRUE)**

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

**tempo.cat <- paste0("\n\n================\n\nERROR IN ", function.name, ": ELEMENT ", i0, " OF mat.list ARGUMENT MUST BE A NUMERIC MATRIX\n\n================\n\n")**

**stop(tempo.cat)**

**}**

**}**

**ident.row.names <- TRUE**

**ident.col.names <- TRUE**

**for(i0 in 2:length(mat.list)){**

**tempo <- fun\_2D\_comp(data1 = mat.list[[1]], data2 = mat.list[[i0]])**

**if(tempo$same.dim == FALSE){**

**tempo.cat <- paste0("\n\n================\n\nERROR IN ", function.name, ": MATRIX ", i0, " OF mat.list ARGUMENT MUST HAVE THE SAME DIMENSION (", paste(dim(mat.list[[i0]]), collapse = " "), ") THAN THE MATRIX 1 IN mat.list (", paste(dim(mat.list[[1]]), collapse = " "), ")\n\n================\n\n")**

**stop(tempo.cat)**

**}**

**if( ! is.null(tempo$same.row.name)){**

**if(tempo$same.row.name != TRUE){** *# != TRUE to deal with NA*

**ident.row.names <- FALSE**

**}**

**}**

**if( ! is.null(tempo$same.col.name)){**

**if(tempo$same.col.name != TRUE){** *# != TRUE to deal with NA*

**ident.col.names <- FALSE**

**}**

**}**

**}**

*# end argument checking without fun\_param\_check()*

*# 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\_param\_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\_param\_check()*

*# end argument checking*

*# main code*

**output <- mat.list[[1]]**

**for(i0 in 2:length(mat.list)){**

**output <- get(kind.of.operation)(output, mat.list[[i0]])**

**}**

**dimnames(output) <- NULL**

**if(ident.row.names == TRUE){**

**rownames(output) <- rownames(mat.list[[1]])**

**}**

**if(ident.col.names == TRUE){**

**colnames(output) <- colnames(mat.list[[1]])**

**}**

**return(output)**

**}**

######## fun\_mat\_inv() #### return the inverse of a square matrix

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

**fun\_mat\_inv <- function(mat){**

*# AIM*

*# return the inverse of a square matrix when solve() cannot*

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

*# fun\_param\_check()*

*# ARGUMENTS:*

*# mat: a square numeric matrix without NULL, NA, Inf or single case (dimension 1, 1) of 0*

*# RETURN*

*# the inversed matrix*

*# EXAMPLES*

*# mat1 = matrix(c(1,1,1,2,1,5,9,8,9), ncol = 3) ; fun\_mat\_inv(mat = mat1) # use solve()*

*# mat1 = matrix(c(0,0,0,0,0,0,0,0,0), ncol = 3) ; fun\_mat\_inv(mat = mat1) # use the trick*

*# mat1 = matrix(c(1,1,1,2,Inf,5,9,8,9), ncol = 3) ; fun\_mat\_inv(mat = mat1)*

*# mat1 = matrix(c(1,1,1,2,NA,5,9,8,9), ncol = 3) ; fun\_mat\_inv(mat = mat1)*

*# mat1 = matrix(c(1,2), ncol = 1) ; fun\_mat\_inv(mat = mat1)*

*# mat1 = matrix(0, ncol = 1) ; fun\_mat\_inv(mat = mat1)*

*# mat1 = matrix(2, ncol = 1) ; fun\_mat\_inv(mat = mat1)*

*# DEBUGGING*

*# mat = matrix(c(1,1,1,2,1,5,9,8,9), ncol = 3) # for function debugging*

*# function name*

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

*# end function name*

*# required function checking*

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

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

**stop(tempo.cat)**

**}**

*# end required function checking*

*# argument checking*

*# argument checking with fun\_param\_check()*

**arg.check <- NULL** *# for function debbuging*

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

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

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

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

**stop()** *# nothing else because print = TRUE by default in fun\_param\_check()*

**}**

*# end argument checking with fun\_param\_check()*

*# argument checking without fun\_param\_check()*

**if(ncol(mat) != nrow(mat)){**

**tempo.cat <- paste0("\n\n================\n\nERROR IN ", function.name, ": mat ARGUMENT MUST BE A SQUARE MATRIX\n\n================\n\n")**

**stop(tempo.cat)**

**}**

**if(any(mat %in% c(Inf, -Inf, NA))){**

**tempo.cat <- paste0("\n\n================\n\nERROR IN ", function.name, ": mat ARGUMENT MUST BE A MATRIX WITHOUT Inf, -Inf OR NA\n\n================\n\n")**

**stop(tempo.cat)**

**}**

**if(all(mat == 0) & ncol(mat) == 1){**

**tempo.cat <- paste0("\n\n================\n\nERROR IN ", function.name, ": mat ARGUMENT CANNOT BE A SQUARE MATRIX MADE OF A SINGLE CASE OF 0\n\n================\n\n")**

**stop(tempo.cat)**

**}**

*# end argument checking without fun\_param\_check()*

*# 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\_param\_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\_param\_check()*

*# end argument checking*

*# main code*

**if(any(grepl(x = try(solve(mat), silent = TRUE)[], pattern = "[Ee]rror"))){**

**tempo <- svd(mat)**

**val.critique <- which(tempo$d < 10^-8)**

**Diag.mod <- diag(1 / tempo$d)**

**for(i in val.critique){**

**Diag.mod[i, i] <- 0**

**}**

**return(tempo$v %\*% Diag.mod %\*% t(tempo$u))**

**}else{**

**return(solve(mat))**

**}**

**}**

######## fun\_mat\_fill() #### fill the empty half part of a symmetric square matrix

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

**fun\_mat\_fill <- function(mat, empty.cell.string = 0, warning.print = TRUE){**

*# AIM*

*# detect the empty half part of a symmetric square matrix (either topleft, topright, bottomleft or bottomright)*

*# fill this empty half part using the other symmetric half part of the matrix*

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

*# fun\_param\_check()*

*# ARGUMENTS:*

*# mat: a numeric or character square matrix with the half part (according to the grand diagonal) filled with NA (any kind of matrix), "0" (character matrix) or 0 (numeric matrix) exclusively (not a mix of 0 and NA in the empty part)*

*# empty.cell.string: a numeric, character or NA (no quotes) indicating what empty cells are filled with*

*# warning.print: logical. Print warning message?*

*# RETURN*

*# if warning.print = FALSE, the filled matrix*

*# if warning.print = TRUE, a list containing:*

*# $mat: the filled matrix*

*# $warnings: the warning messages*

*# a plot verification using fun\_gg\_heatmap() is recommanded*

*# EXAMPLES*

*# mat1 = matrix(c(1,NA,NA,NA, 0,2,NA,NA, NA,3,4,NA, 5,6,7,8), ncol = 4) ; mat1 ; fun\_mat\_fill(mat = mat1, empty.cell.string = NA, warning.print = TRUE) # bottomleft example*

*# mat1 = matrix(c(1,1,1,2, 0,2,3,0, NA,3,0,0, 5,0,0,0), ncol = 4) ; mat1 ; fun\_mat\_fill(mat = mat1, empty.cell.string = NA, warning.print = TRUE) # error example*

*# mat1 = matrix(c(1,1,1,2, 0,2,3,0, NA,3,0,0, 5,0,0,0), ncol = 4) ; mat1 ; fun\_mat\_fill(mat = mat1, empty.cell.string = 0, warning.print = TRUE) # bottomright example*

*# mat1 = matrix(c(1,1,1,2, "a",2,3,NA, "a","a",0,0, "a","a","a",0), ncol = 4) ; mat1 ; fun\_mat\_fill(mat = mat1, empty.cell.string = "a", warning.print = TRUE) # topright example*

*# mat1 = matrix(c(0,0,0,2, 0,0,3,0, 0,3,0,NA, 5,0,0,0), ncol = 4) ; mat1 ; fun\_mat\_fill(mat = mat1, empty.cell.string = 0, warning.print = TRUE) # topleft example*

*# mat1 = matrix(c(0,0,0,2, 0,0,3,0, 0,3,0,0, 5,0,0,0), ncol = 4) ; mat1 ; fun\_mat\_fill(mat = mat1, empty.cell.string = 0, warning.print = TRUE) # error example*

*# DEBUGGING*

*# mat = matrix(c(1,NA,NA,NA, 0,2,NA,NA, NA,3,4,NA, 5,6,7,8), ncol = 4) ; empty.cell.string = NA ; warning.print = TRUE # for function debugging*

*# mat = matrix(c(0,0,0,2, 0,0,3,0, 0,3,0,NA, 5,0,0,0), ncol = 4) ; empty.cell.string = 0 ; warning.print = TRUE # for function debugging # topleft example*

*# function name*

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

*# end function name*

*# required function checking*

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

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

**stop(tempo.cat)**

**}**

*# end required function checking*

*# argument checking*

*# argument checking with fun\_param\_check()*

**arg.check <- NULL** *# for function debbuging*

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

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

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

**tempo <- fun\_param\_check(data = empty.cell.string, class = "vector", na.contain = TRUE, fun.name = function.name) ; eval(ee)**

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

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

**stop()** *# nothing else because print = TRUE by default in fun\_param\_check()*

**}**

*# end argument checking with fun\_param\_check()*

*# argument checking without fun\_param\_check()*

**if(ncol(mat) != nrow(mat)){**

**tempo.cat <- paste0("\n\n================\n\nERROR IN ", function.name, ": mat ARGUMENT MUST BE A SQUARE MATRIX\n\n================\n\n")**

**stop(tempo.cat)**

**}**

**if( ! (mode(mat) %in% c("numeric", "character"))){**

**tempo.cat <- paste0("\n\n================\n\nERROR IN ", function.name, ": mat ARGUMENT MUST BE A NUMERIC OR CHARACTER MATRIX\n\n================\n\n")**

**stop(tempo.cat)**

**}**

**if(nrow(mat) == 1 & ncol(mat) == 1){**

**tempo.cat <- paste0("\n\n================\n\nERROR IN ", function.name, ": mat ARGUMENT CANNOT BE A SQUARE MATRIX MADE OF A SINGLE CASE\n\n================\n\n")**

**stop(tempo.cat)**

**}**

**if(ifelse(is.na(empty.cell.string), ! any(is.na(mat)), ! any(mat == empty.cell.string, na.rm = TRUE))){**

**tempo.cat <- paste0("\n\n================\n\nERROR IN ", function.name, ": mat ARGUMENT MATRIX MUST HAVE CELLS WITH THE EMPTY STRING SPECIFIED IN empty.cell.string ARGUMENT\n\n================\n\n")**

**stop(tempo.cat)**

**}**

*# end argument checking without fun\_param\_check()*

*# 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\_param\_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\_param\_check()*

*# end argument checking*

*# main code*

**list.diag <- vector("list", length = nrow(mat) - 1)**

**for(i0 in 1:(nrow(mat) - 1)){**

**list.diag[[i0]] <- numeric(length = nrow(mat) - i0)**

**}**

**sector <- c("topleft", "topright", "bottomright", "bottomleft")**

**diag.scan <-c(** *# same order as sector. Recover each diag from center to corner*

**"mat[as.matrix(as.data.frame(list(1:(nrow(mat) - i1), (ncol(mat) -i1):1)))]",** *# topleft part*

**"mat[as.matrix(as.data.frame(list(1:(nrow(mat) - i1), (1:ncol(mat))[-(1:i1)])))]",** *# topright part*

**"mat[as.matrix(as.data.frame(list((1 + i1):nrow(mat), ncol(mat):(1 + i1))))]",** *# bottomright part*

**"mat[as.matrix(as.data.frame(list((1 + i1):nrow(mat), 1:(ncol(mat) -i1))))]"** *# bottomleft part*

**)**

*# empty part detection*

**tempo.list.diag <- list.diag**

**empty.sector <- NULL**

**full.sector <- NULL**

**warning <- NULL**

**for(i0 in 1:length(sector)){**

**tempo.list.diag <- list.diag**

**for(i1 in 1:(nrow(mat) - 1)){**

**tempo.list.diag[[i1]] <- eval(parse(text = diag.scan[i0]))**

**if(ifelse(is.na(empty.cell.string), ! all(is.na(tempo.list.diag[[i1]])), ! (all(tempo.list.diag[[i1]] == empty.cell.string, na.rm = TRUE) & ! (is.na(all(tempo.list.diag[[i1]] == empty.cell.string, na.rm = FALSE)))))){** *# I had to add this ! (is.na(all(tempo.list.diag[[i1]] == empty.cell.string, na.rm = FALSE))) because all(tempo.list.diag[[i1]] == empty.cell.string, na.rm = FALSE) gives NA and not FALSE if one NA in tempo.list.diag[[i1]] -> not good for if()*

**full.sector <- c(full.sector, sector[i0])**

**break**

**}**

**}**

**if(i1 == nrow(mat) - 1){**

**if(all(unlist(lapply(tempo.list.diag, FUN = function(x){if(is.na(empty.cell.string)){is.na(x)}else{x == empty.cell.string}})), na.rm = TRUE)){**

**empty.sector <- c(empty.sector, sector[i0])**

**tempo.warning <- paste0("THE EMPTY SECTOR IS ", toupper(sector[i0]))**

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

**}else{**

**tempo.cat <- paste0("\n\n================\n\nERROR IN ", function.name, ": THE ", toupper(sector[i0]), " SECTOR, DETECTED AS EMPTY, IS NOT? DIFFERENT VALUES IN THIS SECTOR:\n", paste(names(table(unlist(tempo.list.diag), useNA = "ifany")), collapse = " "), "\n\n================\n\n")**

**stop(tempo.cat)**

**}**

**}**

**}**

**if(length(empty.sector) == 0){**

**tempo.cat <- paste0("\n\n================\n\nERROR IN ", function.name, ": ACCORDING TO empty.cell.string ARGUMENT (", empty.cell.string, "), mat ARGUMENT MATRIX HAS ZERO EMPTY HALF PART\n\n================\n\n")**

**stop(tempo.cat)**

**}else if(length(empty.sector) > 1){**

**tempo.cat <- paste0("\n\n================\n\nERROR IN ", function.name, ": ACCORDING TO empty.cell.string ARGUMENT (", empty.cell.string, "), mat ARGUMENT MATRIX HAS MORE THAN ONE EMPTY HALF PART (ACCORDING TO THE GRAND DIAGONAL): ", paste(empty.sector, collapse = " "), "\n\n================\n\n")**

**stop(tempo.cat)**

**}else if(any(full.sector %in% empty.sector, na.rm = TRUE)){**

**tempo.cat <- paste0("\n\n================\n\nERROR IN ", function.name, ": THE FUNCTION HAS DETECTED EMPTY AND NON EMPTY HALF PART IN THE SAME SECTOR: ", paste(full.sector[full.sector %in% empty.sector], collapse = " "), "\n\n================\n\n")**

**stop(tempo.cat)**

**}else if(length(empty.sector) + length(full.sector)!= 4){**

**tempo.cat <- paste0("\n\n================\n\nERROR IN ", function.name, ": THE FUNCTION HAS DETECTED MORE OR LESS SECTORS THAN 4:\nEMPTY SECTORS:", paste(empty.sector, collapse = " "), "\nFULL SECTORS:", paste(full.sector, collapse = " "), "\n\n================\n\n")**

**stop(tempo.cat)**

**}**

*# end empty part detection*

*# matrix filling*

**for(i1 in 1:(nrow(mat) - 1)){**

**if(empty.sector == "topleft"){**

**eval(parse(text = paste0(diag.scan[1], " <- ", diag.scan[3])))**

**}else if(empty.sector == "topright"){**

**eval(parse(text = paste0(diag.scan[2], " <- ", diag.scan[4])))**

**}else if(empty.sector == "bottomright"){**

**eval(parse(text = paste0(diag.scan[3], " <- ", diag.scan[1])))**

**}else if(empty.sector == "bottomleft"){**

**eval(parse(text = paste0(diag.scan[4], " <- ", diag.scan[2])))**

**}**

**}**

*# end matrix filling*

**if(warning.print == TRUE){**

**return(list(mat = mat, warnings = warning))**

**}else{**

**return(mat)**

**}**

**}**

######## fun\_consec\_pos\_perm() #### progressively breaks a vector order

**fun\_consec\_pos\_perm <- function(data1, data2 = NULL, n = 20, seed = NULL, count.print = 10, text.print = "", cor.method = "spearman", cor.limit = 0.2, path.lib = NULL){**

*# AIM*

*# reorder the elements of the data1 vector by flipping 2 randomly selected consecutive positions either:*

*# 1) n times (when n is precised) or*

*# 2) until the correlation between data1 and data2 decreases down to the cor.limit (0.2 by default). See cor.limit below to deal with negative correlations*

*# Example of consecutive position flipping: ABCD -> BACD -> BADC, etc.*

*# ARGUMENTS*

*# data1: a vector of at least 2 elements. Must be numeric if data2 is specified*

*# data2: a numeric vector of same length as data1*

*# n: number of times "flipping 2 randomly selected consecutive positions". Ignored if data2 is specified*

*# seed: integer number used by set.seed(). BEWARE: if not NULL, fun\_consec\_pos\_perm() will systematically return the same result when the other parameters keep the same settings*

*# count.print: print a working progress message every count.print loop. If count.print > n, then no message will be printed*

*# text.print: optional message to add to the working progress message every count.print loop*

*# cor.method: correlation method. Either "pearson", "kendall" or "spearman". Ignored if data2 is not specified*

*# cor.limit: a correlation limit (between 0 and 1). If cor(data1, data2) is negative and cor.limit is positive, then -cor.limit is used and the process stops until the correlation between data1 and data2 increases up cor.limit (-0.2 by default). Beware: write a positive cor.limit even if cor(data1, data2) is known to be negative. The function will automately use -cor.limit. Ignored if data2 is not specified*

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

*# REQUIRED PACKAGES*

*# lubridate*

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

*# fun\_param\_check()*

*# fun\_pack\_import()*

*# RETURN*

*# a list containing:*

*# $data: the modified vector*

*# $warnings: potential warning messages (in case of negative correlation when data2 is specified)*

*# $cor: a spearman correlation between the initial positions (1:length(data1) and the final positions if data2 is not specified and the final correlation between data1 and data2 otherwise*

*# EXAMPLES*

*# fun\_consec\_pos\_perm(data1 = LETTERS[1:5], data2 = NULL, n = 20, seed = 1, count.print = 10, text.print = "", cor.method = "spearman", cor.limit = 0.2)*

*# fun\_consec\_pos\_perm(data1 = 101:110, data2 = 21:30, n = 20, seed = 1, count.print = 10, text.print = "", cor.method = "spearman", cor.limit = 0.2)*

*# DEBUGGING*

*# data1 = LETTERS[1:5] ; data2 = NULL ; n = 10 ; seed = NULL ; count.print = 10 ; text.print = "" ; cor.method = "spearman" ; cor.limit = 0.2 ; path.lib = NULL*

*# data1 = LETTERS[1:5] ; data2 = NULL ; n = 10 ; seed = 22 ; count.print = 10 ; text.print = "" ; cor.method = "spearman" ; cor.limit = 0.2 ; path.lib = NULL*

*# data1 = 101:110 ; data2 = 21:30 ; n = 10 ; seed = 22 ; count.print = 10 ; text.print = "" ; cor.method = "spearman" ; cor.limit = 0.2 ; path.lib = NULL*

*# function name*

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

*# end function name*

*# required function checking*

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

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

**stop(tempo.cat)**

**}**

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

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

**stop(tempo.cat)**

**}**

*# end required function checking*

*# argument checking*

*# argument checking without fun\_param\_check()*

**if( ! all(is.vector(data1))){**

**tempo.cat <- paste0("\n\n================\n\nERROR IN ", function.name, ": data1 ARGUMENT MUST BE A VECTOR\n\n================\n\n")**

**stop(tempo.cat)**

**}**

**if(length(data1) < 2){**

**tempo.cat <- paste0("\n\n================\n\nERROR IN ", function.name, ": data1 ARGUMENT MUST BE A VECTOR OF MINIMUM LENGTH 2. HERE IT IS: ", length(data1),"\n\n================\n\n")**

**stop(tempo.cat)**

**}**

*# end argument checking without fun\_param\_check()*

*# argument checking with fun\_param\_check()*

**arg.check <- NULL** *# for function debbuging*

**checked.arg.names <- NULL** *# for function debbuging*

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

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

**tempo <- fun\_param\_check(data = data1, class = "vector", mode = "numeric", fun.name = function.name) ; eval(ee)**

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

**tempo.cat <- paste0("\n\n================\n\nERROR IN ", function.name, ": data1 MUST BE A NUMERIC VECTOR IF data2 ARGUMENT IS SPECIFIED\n\n================\n\n")**

**stop(tempo.cat)**

**}**

**if( ! all(is.vector(data2))){**

**tempo.cat <- paste0("\n\n================\n\nERROR IN ", function.name, ": data2 ARGUMENT MUST BE A VECTOR\n\n================\n\n")**

**stop(tempo.cat)**

**}**

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

**if(length(data1) != length(data2)){**

**tempo.cat <- paste0("\n\n================\n\nERROR IN ", function.name, ": data1 AND data2 MUST BE VECTOR OF SAME LENGTH. HERE IT IS ", length(data1)," AND ", length(data2), "\n\n================\n\n")**

**stop(tempo.cat)**

**}**

**}**

**tempo <- fun\_param\_check(data = n, class = "integer", length = 1, double.as.integer.allowed = TRUE, neg.values = FALSE, fun.name = function.name) ; eval(ee)**

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

**tempo <- fun\_param\_check(data = seed, class = "integer", length = 1, double.as.integer.allowed = TRUE, neg.values = FALSE, fun.name = function.name) ; eval(ee)**

**}**

**tempo <- fun\_param\_check(data = count.print, class = "integer", length = 1, double.as.integer.allowed = TRUE, neg.values = FALSE, fun.name = function.name) ; eval(ee)**

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

**tempo <- fun\_param\_check(data = cor.method, options = c("pearson", "kendall", "spearman"), length =1, fun.name = function.name) ; eval(ee)**

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

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

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

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

**cat(paste0("\n\n============\n\nERROR IN ", function.name, ": \nDIRECTORY PATH INDICATED IN THE path.lib PARAMETER DOES NOT EXISTS: ", path.lib, "\n\n============\n\n"))**

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

**}**

**}**

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

**stop()** *# nothing else because print = TRUE by default in fun\_param\_check()*

**}**

*# end argument checking with fun\_param\_check()*

*# 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\_param\_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\_param\_check()*

*# end argument checking*

*# package checking*

**fun\_pack\_import(req.package = "lubridate", path.lib = path.lib)**

*# end package checking*

*# main code*

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

*# code that does not affect set.seed() in the global environment*

*# as soon as set.seed() or a random function, like runif() or sample(), is used, R creates the .Random.seed in the global environment, that will control the subsequent random operations (vector of number used by the random number generator). The .Random.seed object is systematically stored in the global environment, wherever the environment of set.seed() or runif() execution*

*# thanks to this code, when the function end, .Random.seed recovers the last vector of .Random.seed, whatever random execution performed by the function. Thus, if a set.seed() was performed in the global environment, the reproducibility of successive random operation in this environment will not be impaired*

*# Example:*

*## open a new R session. Then*

*## myfunction <- function (){if(exists(".Random.seed", envir = .GlobalEnv)){old <- .Random.seed ; on.exit(assign(".Random.seed", old, env = .GlobalEnv))} ; set.seed(2) ; runif(1)}*

*## set.seed(1) ; rnorm(1) ; rnorm(1) ; rnorm(1)*

*## rerun the same in a new session but without the previous line*

*## set.seed(1) ; rnorm(1) ; myfunction() ; rnorm(1) ; rnorm(1)*

**if(exists(".Random.seed", envir = .GlobalEnv)){** *# .Random.seed does not exists, it means that no random operation has been performed yet in any R environment*

**tempo.random.seed <- .Random.seed**

**on.exit(assign(".Random.seed", tempo.random.seed, env = .GlobalEnv))**

**}else{**

**on.exit(set.seed(NULL))** *# inactivate set.seed()*

**}**

*# end code that does not affect set.seed() in the global environment*

**set.seed(seed)**

**}**

**ini.date <- Sys.time()**

**ini.time <- as.numeric(ini.date)** *# time of process begin, converted into seconds*

**ini.pos <- 1:length(data1)** *# positions of data1 before permutation loops*

**tempo.pos <- ini.pos** *# positions of data1 that will be modified during loops*

**warnings <- NULL**

*# variable allocation before the loops to save time*

**date.tempo <- Sys.time()**

**time.tempo <- as.numeric(date.tempo)**

**lapse.tempo <- lubridate::seconds\_to\_period(time.tempo - ini.time)**

**pos <- 0** *# position randomly selected*

**tempo.count.print <- count.print** *# for the printing message*

*# end variable allocation before the loops to save time*

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

**if(tempo.count.print > n){**

**tempo.count.print <- n**

**}**

**count <- 0**

**# pos.check <- NULL**

**for(i2 in 1:n){**

**count[1] <- count + 1**

**pos[1] <- sample(x = ini.pos[-length(data1)], size = 1, replace = FALSE)** *# selection of 1 position in initial position, without the last because always up permutation (pos -> pos+1 & pos+1 -> pos)*

**tempo.pos[(pos + 1):pos] <- tempo.pos[pos:(pos + 1)]**

**if(count == tempo.count.print){**

**tempo.count.print[1] <- tempo.count.print + count.print**

**date.tempo[1] <- Sys.time()**

**time.tempo[1] <- as.numeric(date.tempo)**

**lapse.tempo[1] <- lubridate::seconds\_to\_period(time.tempo - ini.time)**

**print(paste0("LOOP ", i2, " / ", n, ifelse(text.print == "", "", paste0(" | ", text.print)), " | TIME SPENT: ", lapse.tempo))**

**}**

**# pos.check <- c(pos.check, pos)**

**}**

**}else{**

**tempo.cor <- cor(x = data1, y = data2, use = "pairwise.complete.obs", method = "spearman")**

**if(tempo.cor < 0){**

**tempo.warnings <- paste0("INITIAL ", toupper(cor.method), " CORRELATION BETWEEN data1 AND data2 HAS BEEN DETECTED AS NEGATIVE: ", tempo.cor, ".THE cor.limit PARAMETER WILL BE SWITCHED TO THE NEGATIVE EQUIVALENT: ", -datacor.limit)**

**warnings <- paste0(warnings, ifelse(is.null(warnings), "", "\n"), tempo.warnings)** *# in fact, abs(tempo.cor) is systematicallu used*

**}**

**if(abs(tempo.cor) < cor.limit){** *# randomize directly all the position to be close to correlation zero*

**for(i1 in 1:5){** *# done 5 times to be sure of the complete randomness*

**tempo.pos <- sample(x = tempo.pos, size = length(tempo.pos), replace = FALSE)**

**}**

**}else{**

**count <- 0**

**while(abs(tempo.cor) > cor.limit){**

**count[1] <- count + 1**

**pos[1] <- sample(x = ini.pos[-length(data1)], size = 1, replace = FALSE)** *# selection of 1 position in initial position, without the last because always up permutation (pos -> pos+1 & pos+1 -> pos)*

**tempo.pos[(pos + 1):pos] <- tempo.pos[pos:(pos + 1)]**

**tempo.cor[1] <- cor(x = data1[tempo.pos], y = data2, use = "pairwise.complete.obs", method = cor.method)**

**if(count == tempo.count.print){**

**tempo.count.print[1] <- tempo.count.print + count.print**

**date.tempo[1] <- Sys.time()**

**time.tempo[1] <- as.numeric(date.tempo)**

**lapse.tempo[1] <- lubridate::seconds\_to\_period(time.tempo - ini.time)**

**print(paste0("LOOP ", count, " / ? (WHILE LOOP) ", ifelse(text.print == "", "", paste0(" | ", text.print)), " PERMUTATION IN data1 | CORRELATION LIMIT: ", cor.limit, " | TEMPO CORRELATION: ", round(tempo.cor, 3), " | TIME SPENT: ", lapse.tempo))**

**}**

**}**

**}**

**}**

**output <- list(data = data1[tempo.pos], warnings = warnings, cor = ifelse(is.null(data2), cor(ini.pos, tempo.pos, method = "spearman"), tempo.cor))**

**return(output)**

**}**

################ Graphics management

*# this order can be used:*

*# fun\_window\_width\_resizing()*

*# fun\_open\_window()*

*# fun\_prior\_plot() # not for ggplot2*

*# plot() or any other plotting*

*# fun\_post\_plot() if fun\_prior\_plot() has been used # not for ggplot2*

*# fun\_close\_specif\_window()*

######## fun\_window\_width\_resizing() #### window width depending on classes to plot

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

**fun\_window\_width\_resizing <- function(class.nb, inches.per.class.nb = 1, ini.window.width = 7, inch.left.space, inch.right.space, boundarie.space = 0.5){**

*# AIM*

*# rescale the width of a window to open depending on the number of classes to plot*

*# can be used for height, considering that it is as if it was a width*

*# this order can be used:*

*# fun\_window\_width\_resizing()*

*# fun\_open\_window()*

*# fun\_prior\_plot() # not for ggplot2*

*# plot() or any other plotting*

*# fun\_post\_plot() if fun\_prior\_plot() has been used # not for ggplot2*

*# fun\_close\_specif\_window()*

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

*# fun\_param\_check()*

*# ARGUMENTS*

*# class.nb: number of class to plot*

*# inches.per.class.nb: number of inches per unit of class.nb. 2 means 2 inches for each boxplot for instance*

*# ini.window.width:initial window width in inches*

*# inch.left.space: left horizontal margin of the figure region (in inches)*

*# inch.right.space: right horizontal margin of the figure region (in inches)*

*# boundarie.space: space between the right and left limits of the plotting region and the plot (0.5 means half a class width)*

*# RETURN*

*# the new window width in inches*

*# EXAMPLES*

*# fun\_window\_width\_resizing(class.nb = 10, inches.per.class.nb = 0.2, ini.window.width = 7, inch.left.space = 1, inch.right.space = 1, boundarie.space = 0.5)*

*# DEBUGGING*

*# class.nb = 10 ; inches.per.class.nb = 0.2 ; ini.window.width = 7 ; inch.left.space = 1 ; inch.right.space = 1 ; boundarie.space = 0.5 # for function debugging*

*# function name*

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

*# end function name*

*# required function checking*

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

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

**stop(tempo.cat)**

**}**

*# end required function checking*

*# argument checking*

**arg.check <- NULL** *# for function debbuging*

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

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

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

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

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

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

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

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

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

**stop()** *# nothing else because print = TRUE by default in fun\_param\_check()*

**}**

*# 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\_param\_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\_param\_check()*

*# end argument checking*

*# main code*

**range.max <- class.nb + boundarie.space** *# the max range of the future plot*

**range.min <- boundarie.space** *# the min range of the future plot*

**window.width <- inch.left.space + inch.right.space + inches.per.class.nb \* (range.max - range.min)**

**return(window.width)**

**}**

######## fun\_open\_window() #### Open a GUI or pdf graphic window

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

**fun\_open\_window <- function(pdf.disp = TRUE, path.fun = "working.dir", pdf.name.file = "graph", width.fun = 7, height.fun = 7, paper = "special", no.pdf.overwrite = TRUE, return.output = FALSE){**

*# AIM*

*# open a pdf or screen (GUI) graphic window*

*# BEWARE: on Linux, use pdf.disp = TRUE, if (GUI) graphic window is not always available, meaning that X is not installed (clusters for instance). Use X11() in R to test if available*

*# this order can be used:*

*# fun\_window\_width\_resizing()*

*# fun\_open\_window()*

*# fun\_prior\_plot() # not for ggplot2*

*# plot() or any other plotting*

*# fun\_post\_plot() if fun\_prior\_plot() has been used # not for ggplot2*

*# fun\_close\_specif\_window()*

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

*# fun\_param\_check()*

*# ARGUMENTS:*

*# pdf.disp: use pdf or not*

*# path.fun: where the pdf is saved. Write "working.dir" if working directory is required (default)*

*# pdf.name.file: name of the pdf file containing the graphs (the .pdf extension is added by the function)*

*# width.fun: width of the windows (in inches)*

*# height.fun: height of the windows (in inches)*

*# paper: paper argument of the pdf function (paper format). Only used for pdf(). Either "a4", "letter", "legal", "us", "executive", "a4r", "USr" or "special". If "special", means that width.fun and height.fun specify the paper size*

*# no.pdf.overwrite: existing pdf can be overwritten ? Only used if pdf.disp = TRUE*

*# return.output: return output ? If TRUE but function not assigned, the output list is displayed*

*# RETURN*

*# a list containing:*

*# $pdf.loc: path of the pdf created*

*# $ini.par: initial par() parameters (to reset in a new graph)*

*# $zone.ini: initial window spliting (to reset in a new graph)*

*# EXAMPLES*

*# fun\_open\_window(pdf.disp = FALSE, path.fun = "C:/Users/Gael/Desktop", pdf.name.file = "graph", width.fun = 7, height.fun = 7, paper = "special", no.pdf.overwrite = TRUE, return.output = TRUE)*

*# DEBUGGING*

*# pdf.disp = TRUE ; path.fun = "C:/Users/Gael/Desktop" ; pdf.name.file = "graphs" ; width.fun = 7 ; height.fun = 7 ; paper = "special" ; no.pdf.overwrite = TRUE ; return.output = TRUE # for function debugging*

*# pdf.disp = TRUE ; path.fun = "/pasteur/homes/gmillot/" ; pdf.name.file = "graphs" ; width.fun = 7 ; height.fun = 7 ; paper = "special" ; no.pdf.overwrite = TRUE ; return.output = TRUE # for function debugging*

*# function name*

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

*# end function name*

*# required function checking*

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

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

**stop(tempo.cat)**

**}**

*# end required function checking*

*# argument checking*

**arg.check <- NULL** *# for function debbuging*

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

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

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

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

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

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

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

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

**tempo <- fun\_param\_check(data = paper, options = c("a4", "letter", "legal", "us", "executive", "a4r", "USr", "special", "A4", "LETTER", "LEGAL", "US"), length = 1, fun.name = function.name) ; eval(ee)**

**tempo <- fun\_param\_check(data =no.pdf.overwrite, class = "logical", length = 1, fun.name = function.name) ; eval(ee)**

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

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

**stop()** *# nothing else because print = TRUE by default in fun\_param\_check()*

**}**

*# 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\_param\_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\_param\_check()*

*# end argument checking*

*# main code*

**if(path.fun == "working.dir"){**

**path.fun <- getwd()**

**}else{**

**if(grepl(x = path.fun, pattern = ".+/$")){**

**path.fun <- substr(path.fun, 1, nchar(path.fun) - 1)** *# remove the last /*

**}**

**if(dir.exists(path.fun) == FALSE){**

**tempo.cat <- paste0("\n\n================\n\nERROR IN ", function.name, ": path.fun ARGUMENT DOES NOT CORRESPOND TO EXISTING DIRECTORY\n\n================\n\n")**

**stop(tempo.cat)**

**}**

**}**

**if(Sys.info()["sysname"] == "Windows"){** *# Note that .Platform$OS.type() only says "unix" for macOS and Linux and "Windows" for Windows*

**open.fail <- NULL**

**windows()**

**ini.par <- par(no.readonly = TRUE)** *# to recover the initial graphical parameters if required (reset). BEWARE: this command alone opens a pdf of GUI window if no window already opened. But here, protected with the code because always a tempo window opened*

**invisible(dev.off())** *# close the new window*

**}else if(Sys.info()["sysname"] == "Linux"){**

**if(pdf.disp == TRUE){**

**if(file.exists(paste0(path.fun, "/recover\_ini\_par.pdf"))){**

**tempo.cat <- paste0("\n\n================\n\nPROBLEM IN fun\_open\_window(): THIS FUNCTION CANNOT BE USED ON LINUX IF A recover\_ini\_par.pdf FILE ALREADY EXISTS HERE: ", paste(path.fun, collapse = " "), "\n\n================\n\n")**

**stop(tempo.cat)**

**}else{**

**pdf(width = width.fun, height = height.fun, file=paste0(path.fun, "/recover\_ini\_par.pdf"), paper = paper)**

**ini.par <- par(no.readonly = TRUE)** *# to recover the initial graphical parameters if required (reset). BEWARE: this command alone opens a pdf of GUI window if no window already opened. But here, protected with the code because always a tempo window opened*

**invisible(dev.off())** *# close the pdf windows*

**file.remove(paste0(path.fun, "/recover\_ini\_par.pdf"))** *# remove the pdf file*

**}**

**}else{**

*# test if X11 can be opened*

**if(file.exists(paste0(getwd(), "/Rplots.pdf"))){**

**tempo.cat <- paste0("\n\n================\n\nERROR IN ", function.name, ": THIS FUNCTION CANNOT BE USED ON LINUX IF A Rplots.pdf FILE ALREADY EXISTS HERE: ", getwd(), "\n\n================\n\n")**

**stop(tempo.cat)**

**}else{**

**open.fail <- suppressWarnings(try(X11(), silent = TRUE))[]** *# try to open a X11 window. If open.fail == NULL, no problem, meaning that the X11 window is opened. If open.fail != NULL, a pdf can be opened here paste0(getwd(), "/Rplots.pdf")*

**if(is.null(open.fail)){**

**ini.par <- par(no.readonly = TRUE)** *# to recover the initial graphical parameters if required (reset). BEWARE: this command alone opens a pdf of GUI window if no window already opened. But here, protected with the code because always a tempo window opened*

**invisible(dev.off())** *# close the new window*

**}else if(file.exists(paste0(getwd(), "/Rplots.pdf"))){**

**file.remove(paste0(getwd(), "/Rplots.pdf"))** *# remove the pdf file*

**tempo.cat <- ("\n\n================\n\nPROBLEM IN fun\_open\_window(): THIS FUNCTION CANNOT OPEN GUI ON LINUX OR NON MACOS UNIX SYSTEM (X GRAPHIC INTERFACE HAS TO BE SET).\nTO OVERCOME THIS, PLEASE SET pdf.disp ARGUMENT TO TRUE AND RERUN\n\n================\n\n")**

**stop(tempo.cat)**

**}**

**}**

**}**

**}else{**

**open.fail <- NULL**

**quartz()**

**ini.par <- par(no.readonly = TRUE)** *# to recover the initial graphical parameters if required (reset). BEWARE: this command alone opens a pdf of GUI window if no window already opened. But here, protected with the code because always a tempo window opened*

**invisible(dev.off())** *# close the new window*

**}**

**zone.ini <- matrix(1, ncol=1)** *# to recover the initial parameters for next figure region when device region split into several figure regions*

**if(pdf.disp == TRUE){**

**pdf.loc <- paste0(path.fun, "/", pdf.name.file, ".pdf")**

**if(file.exists(pdf.loc) == TRUE & no.pdf.overwrite == TRUE){**

**tempo.cat <- paste0("\n\n================\n\nERROR IN ", function.name, ": pdf.loc FILE ALREADY EXISTS AND CANNOT BE OVERWRITTEN DUE TO no.pdf.overwrite ARGUMENT SET TO TRUE: ", pdf.loc, "\n\n================\n\n")**

**stop(tempo.cat)**

**}else{**

**pdf(width = width.fun, height = height.fun, file=pdf.loc, paper = paper)**

**}**

**}else if(pdf.disp == FALSE){**

**pdf.loc <- NULL**

**if(Sys.info()["sysname"] == "Windows"){** *# .Platform$OS.type() only says "unix" for macOS and Linux and "Windows" for Windows*

**windows(width = width.fun, height = height.fun, rescale="fixed")**

**}else if(Sys.info()["sysname"] == "Linux"){**

**if( ! is.null(open.fail)){**

**stop("\n\n================\n\nPROBLEM IN fun\_open\_window(): THIS FUNCTION CANNOT OPEN GUI ON LINUX OR NON MACOS UNIX SYSTEM (X GRAPHIC INTERFACE HAS TO BE SET).\nTO OVERCOME THIS, PLEASE SET pdf.disp ARGUMENT TO TRUE AND RERUN\n\n================\n\n")**

**}else{**

**X11(width = width.fun, height = height.fun)**

**}**

**}else{**

**quartz(width = width.fun, height = height.fun)**

**}**

**}**

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

**output <- list(pdf.loc = pdf.loc, ini.par = ini.par, zone.ini = zone.ini)**

**return(output)**

**}**

**}**

######## fun\_prior\_plot() #### Graph param before plotting

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

**fun\_prior\_plot <- function(param.reinitial = FALSE, xlog.scale = FALSE, ylog.scale = FALSE, remove.label = TRUE, remove.x.axis = TRUE, remove.y.axis = TRUE, std.x.range = TRUE, std.y.range = TRUE, down.space = 1, left.space = 1, up.space = 1, right.space = 1, orient = 1, dist.legend = 3.5, tick.length = 0.5, box.type = "n", amplif.label = 1, amplif.axis = 1, display.extend = FALSE, return.par = FALSE){**

*# AIM*

*# very convenient to erase the axes for post plot axis redrawing using fun\_post\_plot()*

*# reinitialize and set the graphic parameters before plotting*

*# CANNOT be used if no graphic device already opened*

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

*# fun\_param\_check()*

*# ARGUMENTS*

*# param.reinitial: reinitialize graphic parameters before applying the new ones, as defined by the other arguments? Either TRUE or FALSE*

*# xlog.scale: Log scale for the x-axis? Either TRUE or FALSE. If TRUE, erases the x-axis, except legend, for further drawing by fun\_post\_plot()(xlog argument of par())*

*# ylog.scale: Log scale for the y-axis? Either TRUE or FALSE. If TRUE, erases the y-axis, except legend, for further drawing by fun\_post\_plot()(ylog argument of par())*

*# remove.label: remove labels (axis legend) of the two axes? Either TRUE or FALSE (ann argument of par())*

*# remove.x.axis: remove x-axis except legend? Either TRUE or FALSE (control the xaxt argument of par()). Automately set to TRUE if xlog.scale == TRUE*

*# remove.y.axis: remove y-axis except legend? Either TRUE or FALSE (control the yaxt argument of par()). Automately set to TRUE if ylog.scale == TRUE*

*# std.x.range: standard range on the x-axis? TRUE (no range extend) or FALSE (4% range extend). Controls xaxs argument of par() (TRUE is xaxs = "i", FALSE is xaxs = "r")*

*# std.y.range: standard range on the y-axis? TRUE (no range extend) or FALSE (4% range extend). Controls yaxs argument of par() (TRUE is yaxs = "i", FALSE is yaxs = "r")*

*# down.space: lower vertical margin (in inches, mai argument of par())*

*# left.space: left horizontal margin (in inches, mai argument of par())*

*# up.space: upper vertical margin between plot region and grapical window (in inches, mai argument of par())*

*# right.space: right horizontal margin (in inches, mai argument of par())*

*# orient: scale number orientation (las argument of par()). 0, always parallel to the axis; 1, always horizontal; 2, always perpendicular to the axis; 3, always vertical*

*# dist.legend: numeric value that moves axis legends away in inches (first number of mgp argument of par() but in inches thus / 0.2)*

*# tick.length: length of the ticks (1 means complete the distance between the plot region and the axis numbers, 0.5 means half the length, etc. 0 means no tick*

*# box.type: bty argument of par(). Either "o", "l", "7", "c", "u", "]", the resulting box resembles the corresponding upper case letter. A value of "n" suppresses the box*

*# amplif.label: increase or decrease the size of the text in legends*

*# amplif.axis: increase or decrease the size of the scale numbers in axis*

*# display.extend: extend display beyond plotting region? Either TRUE or FALSE (xpd argument of par() without NA)*

*# return.par: return graphic parameter modification?*

*# RETURN*

*# return graphic parameter modification*

*# EXAMPLES*

*# fun\_prior\_plot(param.reinitial = FALSE, xlog.scale = FALSE, ylog.scale = FALSE, remove.label = TRUE, remove.x.axis = TRUE, remove.y.axis = TRUE, std.x.range = TRUE, std.y.range = TRUE, down.space = 1, left.space = 1, up.space = 1, right.space = 1, orient = 1, dist.legend = 4.5, tick.length = 0.5, box.type = "n", amplif.label = 1, amplif.axis = 1, display.extend = FALSE, return.par = FALSE)*

*# DEBUGGING*

*# param.reinitial = FALSE ; xlog.scale = FALSE ; ylog.scale = FALSE ; remove.label = TRUE ; remove.x.axis = TRUE ; remove.y.axis = TRUE ; std.x.range = TRUE ; std.y.range = TRUE ; down.space = 1 ; left.space = 1 ; up.space = 1 ; right.space = 1 ; orient = 1 ; dist.legend = 4.5 ; tick.length = 0.5 ; box.type = "n" ; amplif.label = 1 ; amplif.axis = 1 ; display.extend = FALSE ; return.par = FALSE # for function debugging*

*# function name*

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

*# end function name*

*# required function checking*

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

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

**stop(tempo.cat)**

**}**

*# end required function checking*

*# argument checking*

**arg.check <- NULL** *# for function debbuging*

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

**tempo <- fun\_param\_check(data = box.type, options = c("o", "l", "7", "c", "u", "]", "n"), length = 1, fun.name = function.name) ; eval(ee)**

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

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

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

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

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

**stop()** *# nothing else because print = TRUE by default in fun\_param\_check()*

**}**

*# 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\_param\_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\_param\_check()*

*# end argument checking*

*# main code*

**if(is.null(dev.list())){**

**tempo.cat <- paste0("\n\n================\n\nERROR IN ", function.name, ": THIS FUNCTION CANNOT BE USED IF NO GRAPHIC DEVICE ALREADY OPENED (dev.list() IS CURRENTLY NULL)\n\n================\n\n")**

**stop(tempo.cat)**

**}**

**if(param.reinitial == TRUE){**

**if( ! all(names(dev.cur()) == "null device")){**

**active.wind.nb <- dev.cur()**

**}else{**

**active.wind.nb <- 0**

**}**

**if(Sys.info()["sysname"] == "Windows"){** *# Note that .Platform$OS.type() only says "unix" for macOS and Linux and "Windows" for Windows*

**windows()**

**ini.par <- par(no.readonly = TRUE)** *# to recover the initial graphical parameters if required (reset). BEWARE: this command alone opens a pdf of GUI window if no window already opened. But here, protected with the code because always a tempo window opened*

**invisible(dev.off())** *# close the new window*

**}else if(Sys.info()["sysname"] == "Linux"){**

**if(file.exists(paste0(getwd(), "/Rplots.pdf"))){**

**tempo.cat <- paste0("\n\n================\n\nERROR IN ", function.name, ": THIS FUNCTION CANNOT BE USED ON LINUX WITH param.reinitial SET TO TRUE IF A Rplots.pdf FILE ALREADY EXISTS HERE: ", getwd(), "\n\n================\n\n")**

**stop(tempo.cat)**

**}else{**

**open.fail <- suppressWarnings(try(X11(), silent = TRUE))[]** *# try to open a X11 window. If open.fail == NULL, no problem, meaning that the X11 window is opened. If open.fail != NULL, a pdf can be opened here paste0(getwd(), "/Rplots.pdf")*

**if(is.null(open.fail)){**

**ini.par <- par(no.readonly = TRUE)** *# to recover the initial graphical parameters if required (reset). BEWARE: this command alone opens a pdf of GUI window if no window already opened. But here, protected with the code because always a tempo window opened*

**invisible(dev.off())** *# close the new window*

**}else if(file.exists(paste0(getwd(), "/Rplots.pdf"))){**

**ini.par <- par(no.readonly = TRUE)** *# to recover the initial graphical parameters if required (reset). BEWARE: this command alone opens a pdf of GUI window if no window already opened. But here, protected with the code because always a tempo window opened*

**invisible(dev.off())** *# close the new window*

**file.remove(paste0(getwd(), "/Rplots.pdf"))** *# remove the pdf file*

**}else{**

**tempo.cat <- ("\n\n================\n\nPROBLEM IN fun\_prior\_plot(): THIS FUNCTION CANNOT OPEN GUI ON LINUX OR NON MACOS UNIX SYSTEM (X GRAPHIC INTERFACE HAS TO BE SET).\nTO OVERCOME THIS, PLEASE USE PDF GRAPHIC INTERFACES AND RERUN\n\n================\n\n")**

**stop(tempo.cat)**

**}**

**}**

**}else{** *# macOS*

**quartz()**

**ini.par <- par(no.readonly = TRUE)** *# to recover the initial graphical parameters if required (reset). BEWARE: this command alone opens a pdf of GUI window if no window already opened. But here, protected with the code because always a tempo window opened)*

**invisible(dev.off())** *# close the new window*

**}**

**if( ! all(names(dev.cur()) == "null device")){**

**dev.set(active.wind.nb)** *# go back to the active windows if exists*

**par(ini.par)** *# apply the initial par to current window*

**}**

**}**

**if(remove.x.axis == TRUE){**

**par(xaxt = "n")** *# suppress the y-axis label*

**}else{**

**par(xaxt = "s")**

**}**

**if(remove.y.axis == TRUE){**

**par(yaxt = "n")** *# suppress the y-axis label*

**}else{**

**par(yaxt = "s")**

**}**

**if(std.x.range == TRUE){**

**par(xaxs = "i")**

**}else{**

**par(xaxs = "r")**

**}**

**if(std.y.range == TRUE){**

**par(yaxs = "i")**

**}else{**

**par(yaxs = "r")**

**}**

**par(mai = c(down.space, left.space, up.space, right.space), ann = ! remove.label, las = orient, mgp = c(dist.legend/0.2, 1, 0), xpd = display.extend, bty= box.type, cex.lab = amplif.label, cex.axis = amplif.axis)**

**par(tcl = -par()$mgp[2] \* tick.length)** *# tcl gives the length of the ticks as proportion of line text, knowing that mgp is in text lines. So the main ticks are a 0.5 of the distance of the axis numbers by default. The sign provides the side of the tick (negative for outside of the plot region)*

**if(xlog.scale == TRUE){**

**par(xaxt = "n", xlog = TRUE)** *# suppress the x-axis label*

**}else{**

**par(xlog = FALSE)**

**}**

**if(ylog.scale == TRUE){**

**par(yaxt = "n", ylog = TRUE)** *# suppress the y-axis label*

**}else{**

**par(ylog = FALSE)**

**}**

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

**tempo.par <- par()**

**return(tempo.par)**

**}**

**}**

######## fun\_post\_plot() #### Graph param after plotting

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

**fun\_post\_plot <- function(x.side = 0, x.log.scale = FALSE, x.categ = NULL, x.categ.pos = NULL, x.lab = "", x.axis.magnific = 1.5, x.label.magnific = 1.5, x.dist.legend = 0.5, x.nb.inter.tick = 1, y.side = 0, y.log.scale = FALSE, y.categ = NULL, y.categ.pos = NULL, y.lab = "", y.axis.magnific = 1.5, y.label.magnific = 1.5, y.dist.legend = 0.5, y.nb.inter.tick = 1, text.angle = 90, tick.length = 0.5, sec.tick.length = 0.3, bg.color = NULL, grid.lwd = NULL, grid.col = "white", corner.text = "", magnific.corner.text = 1, just.label.add = FALSE, par.reset = FALSE, custom.par = NULL){**

*# AIM*

*# redesign axis. If x.side = 0, y.side = 0, the function just adds text at topright of the graph and reset par() for next graphics and provides outputs (see below)*

*# provide also positions for legend or additional text on the graph*

*# use fun\_prior\_plot() before this function for initial inactivation of the axis drawings*

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

*# fun\_param\_check()*

*# fun\_open\_window() to reinitialize graph parameters if par.reset = TRUE and custom.par = NULL*

*# ARGUMENTS*

*# x.side: axis at the bottom (1) or top (3) of the region figure. Write 0 for no change*

*# x.log.scale: Log scale for the x-axis? Either TRUE or FALSE*

*# x.categ: character vector representing the classes (levels()) to specify when the x-axis is qualititative(stripchart, boxplot)*

*# x.categ.pos: position of the classes names (numeric vector of identical length than x.categ). If left NULL, this will be 1:length(levels())*

*# x.lab: label of the x-axis. If x.side == 0 and x.lab != "", then x.lab is printed*

*# x.axis.magnific: increase or decrease the value to increase or decrease the size of the x axis numbers. Also control the size of displayed categories*

*# x.label.magnific: increase or decrease the value to increase or decrease the size of the x axis legend*

*# x.dist.legend: increase the number to move x-axis legends away in inches (first number of mgp argument of par() but in inches)*

*# x.nb.inter.tick: number of secondary ticks between main ticks on x-axis (only if not log scale). 0 means no secondary ticks*

*# y.side: axis at the left (2) or right (4) of the region figure. Write 0 for no change*

*# y.log.scale: Log scale for the y-axis? Either TRUE or FALSE*

*# y.categ: classes (levels()) to specify when the y-axis is qualititative(stripchart, boxplot)*

*# y.categ.pos: position of the classes names (numeric vector of identical length than y.categ). If left NULL, this will be 1:length(levels())*

*# y.lab: label of the y-axis. If y.side == 0 and y.lab != "", then y.lab is printed*

*# y.axis.magnific: increase or decrease the value to increase or decrease the size of the y axis numbers. Also control the size of displayed categories*

*# y.label.magnific: increase or decrease the value to increase or decrease the size of the y axis legend*

*# y.dist.legend: increase the number to move y-axis legends away in inches (first number of mgp argument of par() but in inches)*

*# y.nb.inter.tick: number of secondary ticks between main ticks on y-axis (only if not log scale). 0 means non secondary ticks*

*# text.angle: angle of the text when axis is qualitative*

*# tick.length: length of the main ticks (1 means complete the distance between the plot region and the axis numbers, 0.5 means half the length, etc., 0 for no ticks)*

*# sec.tick.length: length of the secondary ticks (1 means complete the distance between the plot region and the axis numbers, 0.5 means half the length, etc., 0 for no ticks)*

*# bg.color: background color of the plot region. NULL for no color. BEWARE: cover/hide an existing plot !*

*# grid.lwd: if non NULL, activate the grid line (specify the line width)*

*# grid.col: grid line color (only if grid.lwd non NULL)*

*# corner.text: text to add at the top right corner of the window*

*# magnific.corner.text: increase or decrease the size of the text*

*# par.reset: to reset all the graphics parameters. BEWARE: TRUE can generate display problems, mainly in graphic devices with multiple figure regions*

*# just.label.add: just add axis labels (legend)? Either TRUE or FALSE. If TRUE, at least (x.side == 0 & x.lab != "") or (y.side == 0 & y.lab != "") must be set to display the corresponding x.lab or y.lab*

*# custom.par: list that provides the parameters that reset all the graphics parameters. BEWARE: if NULL and par.reset == TRUE, the default par() parameters are used*

*# RETURN*

*# a list containing:*

*# $x.mid.left.dev.region: middle of the left margin of the device region, in coordinates of the x-axis*

*# $x.left.dev.region: left side of the left margin (including the potential margin of the device region), in coordinates of the x-axis*

*# $x.mid.right.dev.region: middle of the right margin of the device region, in coordinates of the x-axis*

*# $x.right.dev.region: right side of the right margin (including the potential margin of the device region), in coordinates of the x-axis*

*# $x.mid.left.fig.region: middle of the left margin of the figure region, in coordinates of the x-axis*

*# $x.left.fig.region: left side of the left margin, in coordinates of the x-axis*

*# $x.mid.right.fig.region: middle of the right margin of the figure region, in coordinates of the x-axis*

*# $x.right.fig.region: right side of the right margin, in coordinates of the x-axis*

*# $x.left.plot.region: left side of the plot region, in coordinates of the x-axis*

*# $x.right.plot.region: right side of the plot region, in coordinates of the x-axis*

*# $x.mid.plot.region: middle of the plot region, in coordinates of the x-axis*

*# $y.mid.bottom.dev.region: middle of the bottom margin of the device region, in coordinates of the y-axis*

*# $y.bottom.dev.region: bottom side of the bottom margin (including the potential margin of the device region), in coordinates of the y-axis*

*# $y.mid.top.dev.region: middle of the top margin of the device region, in coordinates of the y-axis*

*# $y.top.dev.region: top side of the top margin (including the potential margin of the device region), in coordinates of the y-axis*

*# $y.mid.bottom.fig.region: middle of the bottom margin of the figure region, in coordinates of the y-axis*

*# $y.bottom.fig.region: bottom of the bottom margin of the figure region, in coordinates of the y-axis*

*# $y.mid.top.fig.region: middle of the top margin of the figure region, in coordinates of the y-axis*

*# $y.top.fig.region: top of the top margin of the figure region, in coordinates of the y-axis*

*# $y.top.plot.region: top of the plot region, in coordinates of the y-axis*

*# $y.bottom.plot.region: bottom of the plot region, in coordinates of the y-axis*

*# $y.mid.plot.region: middle of the plot region, in coordinates of the y-axis*

*# $text: warning text*

*# EXAMPLES*

*# Example of log axis with log y-axis and unmodified x-axis:*

*# prior.par <- fun\_prior\_plot(param.reinitial = TRUE, xlog.scale = FALSE, ylog.scale = TRUE, remove.label = TRUE, remove.x.axis = FALSE, remove.y.axis = TRUE, down.space = 1, left.space = 1, up.space = 1, right.space = 1, orient = 1, dist.legend = 0.5, tick.length = 0.5, box.type = "n", amplif.label = 1, amplif.axis = 1, display.extend = FALSE, return.par = TRUE) ; plot(1:100, log = "y") ; fun\_post\_plot(y.side = 2, y.log.scale = prior.par$ylog, x.lab = "Values", y.lab = "TEST", y.axis.magnific = 1.25, y.label.magnific = 1.5, y.dist.legend = 0.7, just.label.add = ! prior.par$ann)*

*# Example of log axis with redrawn x-axis and y-axis:*

*# prior.par <- fun\_prior\_plot(param.reinitial = TRUE) ; plot(1:100) ; fun\_post\_plot(x.side = 1, x.lab = "Values", y.side = 2, y.lab = "TEST", y.axis.magnific = 1, y.label.magnific = 2, y.dist.legend = 0.6)*

*# example with margins in the device region:*

*# windows(5,5) ; par(mai=c(0.5,0.5,0.5,0.5), omi = c(0.25,0.25,1,0.25), xaxs = "i", yaxs = "i") ; plot(0:10) ; a <- fun\_post\_plot(x.side = 0, y.side = 0) ; x <- c(a$x.mid.left.dev.region, a$x.left.dev.region, a$x.mid.right.dev.region, a$x.right.dev.region, a$x.mid.left.fig.region, a$x.left.fig.region, a$x.mid.right.fig.region, a$x.right.fig.region, a$x.right.plot.region, a$x.left.plot.region, a$x.mid.plot.region) ; y <- c(a$y.mid.bottom.dev.region, a$y.bottom.dev.region, a$y.mid.top.dev.region, a$y.top.dev.region, a$y.mid.bottom.fig.region, a$y.bottom.fig.region, a$y.mid.top.fig.region, a$y.top.fig.region, a$y.top.plot.region, a$y.bottom.plot.region, a$y.mid.plot.region) ; par(xpd = NA) ; points(x = rep(5, length(y)), y = y, pch = 16, col = "red") ; text(x = rep(5, length(y)), y = y, c("y.mid.bottom.dev.region", "y.bottom.dev.region", "y.mid.top.dev.region", "y.top.dev.region", "y.mid.bottom.fig.region", "y.bottom.fig.region", "y.mid.top.fig.region", "y.top.fig.region", "y.top.plot.region", "y.bottom.plot.region", "y.mid.plot.region"), cex = 0.65, col = grey(0.25)) ; points(y = rep(5, length(x)), x = x, pch = 16, col = "blue") ; text(y = rep(5, length(x)), x = x, c("x.mid.left.dev.region", "x.left.dev.region", "x.mid.right.dev.region", "x.right.dev.region", "x.mid.left.fig.region", "x.left.fig.region", "x.mid.right.fig.region", "x.right.fig.region", "x.right.plot.region", "x.left.plot.region", "x.mid.plot.region"), cex = 0.65, srt = 90, col = grey(0.25))*

*# DEBUGGING*

*# x.side = 0 ; x.log.scale = FALSE ; x.categ = NULL ; x.categ.pos = NULL ; x.lab = "" ; x.axis.magnific = 1.5 ; x.label.magnific = 1.5 ; x.dist.legend = 1 ; x.nb.inter.tick = 1 ; y.side = 0 ; y.log.scale = FALSE ; y.categ = NULL ; y.categ.pos = NULL ; y.lab = "" ; y.axis.magnific = 1.5 ; y.label.magnific = 1.5 ; y.dist.legend = 0.7 ; y.nb.inter.tick = 1 ; text.angle = 90 ; tick.length = 0.5 ; sec.tick.length = 0.3 ; bg.color = NULL ; grid.lwd = NULL ; grid.col = "white" ; corner.text = "" ; magnific.corner.text = 1 ; just.label.add = FALSE ; par.reset = FALSE ; custom.par = NULL # for function debugging*

*# function name*

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

*# end function name*

*# required function checking*

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

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

**stop(tempo.cat)**

**}**

**if(length(find("fun\_open\_window", mode = "function")) == 0){**

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

**stop(tempo.cat)**

**}**

*# end required function checking*

*# argument checking*

**arg.check <- NULL** *# for function debbuging*

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

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

**tempo <- fun\_param\_check(data = x.side, options = c(0, 1, 3), length = 1, fun.name = function.name) ; eval(ee)**

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

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

**tempo <- fun\_param\_check(data = x.categ, class = "character", na.contain = TRUE, fun.name = function.name) ; eval(ee)**

**}**

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

**tempo <- fun\_param\_check(data = x.categ.pos, class = "vector", mode = "numeric", fun.name = function.name) ; eval(ee)**

**}**

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

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

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

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

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

**tempo <- fun\_param\_check(data = y.side, options = c(0, 2, 4), length = 1, fun.name = function.name) ; eval(ee)**

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

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

**tempo <- fun\_param\_check(data = y.categ, class = "character", na.contain = TRUE, fun.name = function.name) ; eval(ee)**

**}**

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

**tempo <- fun\_param\_check(data = y.categ.pos, class = "vector", mode = "numeric", fun.name = function.name) ; eval(ee)**

**}**

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

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

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

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

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

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

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

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

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

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

**if( ! (bg.color %in% colors() | grepl(pattern = "^#", bg.color))){** *# check color*

**tempo.cat <- paste0("\n\n================\n\nERROR IN ", function.name, ": bg.color ARGUMENT MUST BE A HEXADECIMAL COLOR VECTOR STARTING BY # OR A COLOR NAME GIVEN BY colors()\n\n================\n\n")**

**stop(tempo.cat)**

**}**

**}**

**if( ! is.null(grid.lwd)){**

**tempo <- fun\_param\_check(data = grid.lwd, class = "vector", mode = "numeric", neg.values = FALSE, fun.name = function.name) ; eval(ee)**

**}**

**if( ! is.null(grid.col)){**

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

**if( ! (grid.col %in% colors() | grepl(pattern = "^#", grid.col))){** *# check color*

**tempo.cat <- paste0("\n\n================\n\nERROR IN ", function.name, ": grid.col ARGUMENT MUST BE A HEXADECIMAL COLOR VECTOR STARTING BY # OR A COLOR NAME GIVEN BY colors()\n\n================\n\n")**

**stop(tempo.cat)**

**}**

**}**

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

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

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

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

**if( ! is.null(custom.par)){**

**tempo <- fun\_param\_check(data = custom.par, typeof = "list", length = 1, fun.name = function.name) ; eval(ee)**

**}**

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

**stop()** *# nothing else because print = TRUE by default in fun\_param\_check()*

**}**

*# 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\_param\_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\_param\_check()*

*# end argument checking*

*# main code*

**text <- NULL**

**par(tcl = -par()$mgp[2] \* tick.length)**

**if(x.log.scale == TRUE){**

**grid.coord.x <- c(10^par("usr")[1], 10^par("usr")[2])**

**}else{**

**grid.coord.x <- c(par("usr")[1], par("usr")[2])**

**}**

**if(y.log.scale == TRUE){**

**grid.coord.y <- c(10^par("usr")[3], 10^par("usr")[4])**

**}else{**

**grid.coord.y <- c(par("usr")[3], par("usr")[4])**

**}**

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

**rect(grid.coord.x[1], grid.coord.y[1], grid.coord.x[2], grid.coord.y[2], col = bg.color, border = NA)**

**}**

**if( ! is.null(grid.lwd)){**

**grid(nx = NA, ny = NULL, col = grid.col, lty = 1, lwd = grid.lwd)**

**}**

**if(x.log.scale == TRUE){**

**x.mid.left.dev.region <- 10^(par("usr")[1] - ((par("usr")[2] - par("usr")[1]) / (par("plt")[2] - par("plt")[1])) \* par("plt")[1] - ((par("usr")[2] - par("usr")[1]) / ((par("omd")[2] - par("omd")[1]) \* (par("plt")[2] - par("plt")[1]))) \* par("omd")[1] / 2)** *# in x coordinates, to position axis labeling at the bottom of the graph (according to x scale)*

**x.left.dev.region <- 10^(par("usr")[1] - ((par("usr")[2] - par("usr")[1]) / (par("plt")[2] - par("plt")[1])) \* par("plt")[1] - ((par("usr")[2] - par("usr")[1]) / ((par("omd")[2] - par("omd")[1]) \* (par("plt")[2] - par("plt")[1]))) \* par("omd")[1])** *# in x coordinates*

**x.mid.right.dev.region <- 10^(par("usr")[2] + ((par("usr")[2] - par("usr")[1]) / (par("plt")[2] - par("plt")[1])) \* (1 - par("plt")[2]) + ((par("usr")[2] - par("usr")[1]) / ((par("omd")[2] - par("omd")[1]) \* (par("plt")[2] - par("plt")[1]))) \* (1 - par("omd")[2]) / 2)** *# in x coordinates, to position axis labeling at the top of the graph (according to x scale)*

**x.right.dev.region <- 10^(par("usr")[2] + ((par("usr")[2] - par("usr")[1]) / (par("plt")[2] - par("plt")[1])) \* (1 - par("plt")[2]) + ((par("usr")[2] - par("usr")[1]) / ((par("omd")[2] - par("omd")[1]) \* (par("plt")[2] - par("plt")[1]))) \* (1 - par("omd")[2]))** *# in x coordinates*

**x.mid.left.fig.region <- 10^(par("usr")[1] - ((par("usr")[2] - par("usr")[1]) / (par("plt")[2] - par("plt")[1])) \* par("plt")[1] / 2)** *# in x coordinates, to position axis labeling at the bottom of the graph (according to x scale)*

**x.left.fig.region <- 10^(par("usr")[1] - ((par("usr")[2] - par("usr")[1]) / (par("plt")[2] - par("plt")[1])) \* par("plt")[1])** *# in x coordinates*

**x.mid.right.fig.region <- 10^(par("usr")[2] + ((par("usr")[2] - par("usr")[1]) / (par("plt")[2] - par("plt")[1])) \* (1 - par("plt")[2]) / 2)** *# in x coordinates, to position axis labeling at the top of the graph (according to x scale)*

**x.right.fig.region <- 10^(par("usr")[2] + ((par("usr")[2] - par("usr")[1]) / (par("plt")[2] - par("plt")[1])) \* (1 - par("plt")[2]))** *# in x coordinates*

**x.left.plot.region <- 10^par("usr")[1]** *# in x coordinates, left of the plot region (according to x scale)*

**x.right.plot.region <- 10^par("usr")[2]** *# in x coordinates, right of the plot region (according to x scale)*

**x.mid.plot.region <- 10^((par("usr")[2] + par("usr")[1]) / 2)** *# in x coordinates, right of the plot region (according to x scale)*

**}else{**

**x.mid.left.dev.region <- (par("usr")[1] - ((par("usr")[2] - par("usr")[1]) / (par("plt")[2] - par("plt")[1])) \* par("plt")[1] - ((par("usr")[2] - par("usr")[1]) / ((par("omd")[2] - par("omd")[1]) \* (par("plt")[2] - par("plt")[1]))) \* par("omd")[1] / 2)** *# in x coordinates, to position axis labeling at the bottom of the graph (according to x scale)*

**x.left.dev.region <- (par("usr")[1] - ((par("usr")[2] - par("usr")[1]) / (par("plt")[2] - par("plt")[1])) \* par("plt")[1] - ((par("usr")[2] - par("usr")[1]) / ((par("omd")[2] - par("omd")[1]) \* (par("plt")[2] - par("plt")[1]))) \* par("omd")[1])** *# in x coordinates*

**x.mid.right.dev.region <- (par("usr")[2] + ((par("usr")[2] - par("usr")[1]) / (par("plt")[2] - par("plt")[1])) \* (1 - par("plt")[2]) + ((par("usr")[2] - par("usr")[1]) / ((par("omd")[2] - par("omd")[1]) \* (par("plt")[2] - par("plt")[1]))) \* (1 - par("omd")[2]) / 2)** *# in x coordinates, to position axis labeling at the top of the graph (according to x scale)*

**x.right.dev.region <- (par("usr")[2] + ((par("usr")[2] - par("usr")[1]) / (par("plt")[2] - par("plt")[1])) \* (1 - par("plt")[2]) + ((par("usr")[2] - par("usr")[1]) / ((par("omd")[2] - par("omd")[1]) \* (par("plt")[2] - par("plt")[1]))) \* (1 - par("omd")[2]))** *# in x coordinates*

**x.mid.left.fig.region <- (par("usr")[1] - ((par("usr")[2] - par("usr")[1]) / (par("plt")[2] - par("plt")[1])) \* par("plt")[1] / 2)** *# in x coordinates, to position axis labeling at the bottom of the graph (according to x scale)*

**x.left.fig.region <- (par("usr")[1] - ((par("usr")[2] - par("usr")[1]) / (par("plt")[2] - par("plt")[1])) \* par("plt")[1])** *# in x coordinates*

**x.mid.right.fig.region <- (par("usr")[2] + ((par("usr")[2] - par("usr")[1]) / (par("plt")[2] - par("plt")[1])) \* (1 - par("plt")[2]) / 2)** *# in x coordinates, to position axis labeling at the top of the graph (according to x scale)*

**x.right.fig.region <- (par("usr")[2] + ((par("usr")[2] - par("usr")[1]) / (par("plt")[2] - par("plt")[1])) \* (1 - par("plt")[2]))** *# in x coordinates*

**x.left.plot.region <- par("usr")[1]** *# in x coordinates, left of the plot region (according to x scale)*

**x.right.plot.region <- par("usr")[2]** *# in x coordinates, right of the plot region (according to x scale)*

**x.mid.plot.region <- (par("usr")[2] + par("usr")[1]) / 2** *# in x coordinates, right of the plot region (according to x scale)*

**}**

**if(y.log.scale == TRUE){**

**y.mid.bottom.dev.region <- 10^(par("usr")[3] - ((par("usr")[4] - par("usr")[3]) / (par("plt")[4] - par("plt")[3])) \* par("plt")[3] - ((par("usr")[4] - par("usr")[3]) / ((par("omd")[4] - par("omd")[3]) \* (par("plt")[4] - par("plt")[3]))) \* (par("omd")[3] / 2))** *# in y coordinates, to position axis labeling at the bottom of the graph (according to y scale). Ex mid.bottom.space*

**y.bottom.dev.region <- 10^(par("usr")[3] - ((par("usr")[4] - par("usr")[3]) / (par("plt")[4] - par("plt")[3])) \* par("plt")[3] - ((par("usr")[4] - par("usr")[3]) / ((par("omd")[4] - par("omd")[3]) \* (par("plt")[4] - par("plt")[3]))) \* par("omd")[3])** *# in y coordinates*

**y.mid.top.dev.region <- 10^(par("usr")[4] + ((par("usr")[4] - par("usr")[3]) / (par("plt")[4] - par("plt")[3])) \* (1 - par("plt")[4]) + ((par("usr")[4] - par("usr")[3]) / ((par("omd")[4] - par("omd")[3]) \* (par("plt")[4] - par("plt")[3]))) \* (1 - par("omd")[4]) / 2)** *# in y coordinates, to position axis labeling at the top of the graph (according to y scale). Ex mid.top.space*

**y.top.dev.region <- 10^(par("usr")[4] + ((par("usr")[4] - par("usr")[3]) / (par("plt")[4] - par("plt")[3])) \* (1 - par("plt")[4]) + ((par("usr")[4] - par("usr")[3]) / ((par("omd")[4] - par("omd")[3]) \* (par("plt")[4] - par("plt")[3]))) \* (1 - par("omd")[4]))** *# in y coordinates*

**y.mid.bottom.fig.region <- 10^(par("usr")[3] - ((par("usr")[4] - par("usr")[3]) / (par("plt")[4] - par("plt")[3])) \* par("plt")[3] / 2)** *# in y coordinates, to position axis labeling at the bottom of the graph (according to y scale). Ex mid.bottom.space*

**y.bottom.fig.region <- 10^(par("usr")[3] - ((par("usr")[4] - par("usr")[3]) / (par("plt")[4] - par("plt")[3])) \* par("plt")[3])** *# in y coordinates*

**y.mid.top.fig.region <- 10^(par("usr")[4] + ((par("usr")[4] - par("usr")[3]) / (par("plt")[4] - par("plt")[3])) \* (1 - par("plt")[4]) / 2)** *# in y coordinates, to position axis labeling at the top of the graph (according to y scale). Ex mid.top.space*

**y.top.fig.region <- 10^(par("usr")[4] + ((par("usr")[4] - par("usr")[3]) / (par("plt")[4] - par("plt")[3])) \* (1 - par("plt")[4]))** *# in y coordinates*

**y.top.plot.region <- 10^par("usr")[4]** *# in y coordinates, top of the plot region (according to y scale)*

**y.bottom.plot.region <- 10^par("usr")[3]** *# in y coordinates, bottom of the plot region (according to y scale)*

**y.mid.plot.region <- (par("usr")[3] + par("usr")[4]) / 2** *# in x coordinates, right of the plot region (according to x scale)*

**}else{**

**y.mid.bottom.dev.region <- (par("usr")[3] - ((par("usr")[4] - par("usr")[3]) / (par("plt")[4] - par("plt")[3])) \* par("plt")[3] - ((par("usr")[4] - par("usr")[3]) / ((par("omd")[4] - par("omd")[3]) \* (par("plt")[4] - par("plt")[3]))) \* (par("omd")[3] / 2))** *# in y coordinates, to position axis labeling at the bottom of the graph (according to y scale). Ex mid.bottom.space*

**y.bottom.dev.region <- (par("usr")[3] - ((par("usr")[4] - par("usr")[3]) / (par("plt")[4] - par("plt")[3])) \* par("plt")[3] - ((par("usr")[4] - par("usr")[3]) / ((par("omd")[4] - par("omd")[3]) \* (par("plt")[4] - par("plt")[3]))) \* par("omd")[3])** *# in y coordinates*

**y.mid.top.dev.region <- (par("usr")[4] + ((par("usr")[4] - par("usr")[3]) / (par("plt")[4] - par("plt")[3])) \* (1 - par("plt")[4]) + ((par("usr")[4] - par("usr")[3]) / ((par("omd")[4] - par("omd")[3]) \* (par("plt")[4] - par("plt")[3]))) \* (1 - par("omd")[4]) / 2)** *# in y coordinates, to position axis labeling at the top of the graph (according to y scale). Ex mid.top.space*

**y.top.dev.region <- (par("usr")[4] + ((par("usr")[4] - par("usr")[3]) / (par("plt")[4] - par("plt")[3])) \* (1 - par("plt")[4]) + ((par("usr")[4] - par("usr")[3]) / ((par("omd")[4] - par("omd")[3]) \* (par("plt")[4] - par("plt")[3]))) \* (1 - par("omd")[4]))** *# in y coordinates*

**y.mid.bottom.fig.region <- (par("usr")[3] - ((par("usr")[4] - par("usr")[3]) / (par("plt")[4] - par("plt")[3])) \* par("plt")[3] / 2)** *# in y coordinates, to position axis labeling at the bottom of the graph (according to y scale). Ex mid.bottom.space*

**y.bottom.fig.region <- (par("usr")[3] - ((par("usr")[4] - par("usr")[3]) / (par("plt")[4] - par("plt")[3])) \* par("plt")[3])** *# in y coordinates*

**y.mid.top.fig.region <- (par("usr")[4] + ((par("usr")[4] - par("usr")[3]) / (par("plt")[4] - par("plt")[3])) \* (1 - par("plt")[4]) / 2)** *# in y coordinates, to position axis labeling at the top of the graph (according to y scale). Ex mid.top.space*

**y.top.fig.region <- (par("usr")[4] + ((par("usr")[4] - par("usr")[3]) / (par("plt")[4] - par("plt")[3])) \* (1 - par("plt")[4]))** *# in y coordinates*

**y.top.plot.region <- par("usr")[4]** *# in y coordinates, top of the plot region (according to y scale)*

**y.bottom.plot.region <- par("usr")[3]** *# in y coordinates, bottom of the plot region (according to y scale)*

**y.mid.plot.region <- ((par("usr")[3] + par("usr")[4]) / 2)** *# in x coordinates, right of the plot region (according to x scale)*

**}**

**if(x.side == 1 | x.side == 3){**

**par(xpd=FALSE, xaxt="s")**

**if(is.null(x.categ) & x.log.scale == TRUE){**

**if(any(par()$xaxp[1:2] == 0)){**

**if(par()$xaxp[1] == 0){**

**par(xaxp = c(10^-30, par()$xaxp[2:3]))** *# because log10(par()$xaxp[1] == 0) == -Inf*

**}**

**if(par()$xaxp[2] == 0){**

**par(xaxp = c(par()$xaxp[1], 10^-30, par()$xaxp[3]))** *# because log10(par()$xaxp[2] == 0) == -Inf*

**}**

**}**

**axis(side=x.side, at=c(10^par()$usr[1], 10^par()$usr[2]), labels=rep("", 2), lwd=1, lwd.ticks=0)** *# draw the axis line*

**mtext(side = x.side, text = x.lab, line = x.dist.legend / 0.2, las = 0, cex = x.label.magnific)**

**par(tcl = -par()$mgp[2] \* sec.tick.length)** *# length of the secondary ticks are reduced*

**suppressWarnings(rug(10^outer(c((log10(par("xaxp")[1]) -1):log10(par("xaxp")[2])), log10(1:10), "+"), ticksize = NA, side = x.side))** *# ticksize = NA to allow the use of par()$tcl value*

**par(tcl = -par()$mgp[2] \* tick.length)** *# back to main ticks*

**axis(side = x.side, at = c(1e-15, 1e-14, 1e-13, 1e-12, 1e-11, 1e-10, 1e-9, 1e-8, 1e-7, 1e-6, 1e-5, 1e-4, 1e-3, 1e-2, 1e-1, 1e0, 1e1, 1e2, 1e3, 1e4, 1e5, 1e6, 1e7, 1e8, 1e9, 1e10), labels = expression(10^-15, 10^-14, 10^-13, 10^-12, 10^-11, 10^-10, 10^-9, 10^-8, 10^-7, 10^-6, 10^-5, 10^-4, 10^-3, 10^-2, 10^-1, 10^0, 10^1, 10^2, 10^3, 10^4, 10^5, 10^6, 10^7, 10^8, 10^9, 10^10), lwd = 0, lwd.ticks = 1, cex.axis = x.axis.magnific)**

**x.text <- 10^par("usr")[2]**

**}else if(is.null(x.categ) & x.log.scale == FALSE){**

**axis(side=x.side, at=c(par()$usr[1], par()$usr[2]), labels=rep("", 2), lwd=1, lwd.ticks=0)** *# draw the axis line*

**axis(side=x.side, at=round(seq(par()$xaxp[1], par()$xaxp[2], length.out=par()$xaxp[3]+1), 2), cex.axis = x.axis.magnific)** *# axis(side=x.side, at=round(seq(par()$xaxp[1], par()$xaxp[2], length.out=par()$xaxp[3]+1), 2), labels = format(round(seq(par()$xaxp[1], par()$xaxp[2], length.out=par()$xaxp[3]+1), 2), big.mark=','), cex.axis = x.axis.magnific) # to get the 1000 comma separator*

**mtext(side = x.side, text = x.lab, line = x.dist.legend / 0.2, las = 0, cex = x.label.magnific)**

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

**inter.tick.unit <- (par("xaxp")[2] - par("xaxp")[1]) / par("xaxp")[3]**

**par(tcl = -par()$mgp[2] \* sec.tick.length)** *# length of the ticks are reduced*

**suppressWarnings(rug(seq(par("xaxp")[1] - 10 \* inter.tick.unit, par("xaxp")[2] + 10 \* inter.tick.unit, by = inter.tick.unit / (1 + x.nb.inter.tick)), ticksize = NA, x.side))** *# ticksize = NA to allow the use of par()$tcl value*

**par(tcl = -par()$mgp[2] \* tick.length)** *# back to main ticks*

**}**

**x.text <- par("usr")[2]**

**}else if(( ! is.null(x.categ)) & x.log.scale == FALSE){**

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

**x.categ.pos <- 1:length(x.categ)**

**}else if(length(x.categ.pos) != length(x.categ)){**

**stop("\n\nPROBLEM: x.categ.pos MUST BE THE SAME LENGTH AS x.categ\n\n")**

**}**

**par(xpd = TRUE)**

**if(x.side == 1){**

**segments(x0 = x.left.plot.region, x1 = x.right.plot.region, y0 = y.bottom.plot.region, y1 = y.bottom.plot.region)** *# draw the line of the axis*

**text(x = x.categ.pos, y = y.mid.bottom.fig.region, labels = x.categ, srt = text.angle, cex = x.axis.magnific)**

**}else if(x.side == 3){**

**segments(x0 = x.left.plot.region, x1 = x.right.plot.region, y0 = y.top.plot.region, y1 = y.top.plot.region)** *# draw the line of the axis*

**text(x = x.categ.pos, y = y.mid.top.fig.region, labels = x.categ, srt = text.angle, cex = x.axis.magnific)**

**}else{**

**stop("\n\nARGUMENT x.side CAN ONLY BE 1 OR 3\n\n")**

**}**

**par(xpd = FALSE)**

**x.text <- par("usr")[2]**

**}else{**

**stop("\n\nPROBLEM WITH THE x.side (", x.side ,") OR x.log.scale (", x.log.scale,") ARGUMENTS\n\n")**

**}**

**}else{**

**x.text <- par("usr")[2]**

**}**

**if(y.side == 2 | y.side == 4){**

**par(xpd=FALSE, yaxt="s")**

**if(is.null(y.categ) & y.log.scale == TRUE){**

**if(any(par()$yaxp[1:2] == 0)){**

**if(par()$yaxp[1] == 0){**

**par(yaxp = c(10^-30, par()$yaxp[2:3]))** *# because log10(par()$yaxp[1] == 0) == -Inf*

**}**

**if(par()$yaxp[2] == 0){**

**par(yaxp = c(par()$yaxp[1], 10^-30, par()$yaxp[3]))** *# because log10(par()$yaxp[2] == 0) == -Inf*

**}**

**}**

**axis(side=y.side, at=c(10^par()$usr[3], 10^par()$usr[4]), labels=rep("", 2), lwd=1, lwd.ticks=0)** *# draw the axis line*

**par(tcl = -par()$mgp[2] \* sec.tick.length)** *# length of the ticks are reduced*

**suppressWarnings(rug(10^outer(c((log10(par("yaxp")[1])-1):log10(par("yaxp")[2])), log10(1:10), "+"), ticksize = NA, side = y.side))** *# ticksize = NA to allow the use of par()$tcl value*

**par(tcl = -par()$mgp[2] \* tick.length)** *# back to main tick length*

**axis(side = y.side, at = c(1e-15, 1e-14, 1e-13, 1e-12, 1e-11, 1e-10, 1e-9, 1e-8, 1e-7, 1e-6, 1e-5, 1e-4, 1e-3, 1e-2, 1e-1, 1e0, 1e1, 1e2, 1e3, 1e4, 1e5, 1e6, 1e7, 1e8, 1e9, 1e10), labels = expression(10^-15, 10^-14, 10^-13, 10^-12, 10^-11, 10^-10, 10^-9, 10^-8, 10^-7, 10^-6, 10^-5, 10^-4, 10^-3, 10^-2, 10^-1, 10^0, 10^1, 10^2, 10^3, 10^4, 10^5, 10^6, 10^7, 10^8, 10^9, 10^10), lwd = 0, lwd.ticks = 1, cex.axis = y.axis.magnific)**

**y.text <- 10^(par("usr")[4] + (par("usr")[4] - par("usr")[3]) / (par("plt")[4] - par("plt")[3]) \* (1 - par("plt")[4]))**

**mtext(side = y.side, text = y.lab, line = y.dist.legend / 0.2, las = 0, cex = y.label.magnific)**

**}else if(is.null(y.categ) & y.log.scale == FALSE){**

**axis(side=y.side, at=c(par()$usr[3], par()$usr[4]), labels=rep("", 2), lwd=1, lwd.ticks=0)** *# draw the axis line*

**axis(side=y.side, at=round(seq(par()$yaxp[1], par()$yaxp[2], length.out=par()$yaxp[3]+1), 2), cex.axis = y.axis.magnific)**

**mtext(side = y.side, text = y.lab, line = y.dist.legend / 0.2, las = 0, cex = y.label.magnific)**

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

**inter.tick.unit <- (par("yaxp")[2] - par("yaxp")[1]) / par("yaxp")[3]**

**par(tcl = -par()$mgp[2] \* sec.tick.length)** *# length of the ticks are reduced*

**suppressWarnings(rug(seq(par("yaxp")[1] - 10 \* inter.tick.unit, par("yaxp")[2] + 10 \* inter.tick.unit, by = inter.tick.unit / (1 + y.nb.inter.tick)), ticksize = NA, side=y.side))** *# ticksize = NA to allow the use of par()$tcl value*

**par(tcl = -par()$mgp[2] \* tick.length)** *# back to main tick length*

**}**

**y.text <- (par("usr")[4] + (par("usr")[4] - par("usr")[3]) / (par("plt")[4] - par("plt")[3]) \* (1 - par("plt")[4]))**

**}else if(( ! is.null(y.categ)) & y.log.scale == FALSE){**

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

**y.categ.pos <- 1:length(y.categ)**

**}else if(length(y.categ.pos) != length(y.categ)){**

**stop("\n\nPROBLEM: y.categ.pos MUST BE THE SAME LENGTH AS y.categ\n\n")**

**}**

**axis(side = y.side, at = y.categ.pos, labels = rep("", length(y.categ)), lwd=0, lwd.ticks=1)** *# draw the line of the axis*

**par(xpd = TRUE)**

**if(y.side == 2){**

**text(x = x.mid.left.fig.region, y = y.categ.pos, labels = y.categ, srt = text.angle, cex = y.axis.magnific)**

**}else if(y.side == 4){**

**text(x = x.mid.right.fig.region, y = y.categ.pos, labels = y.categ, srt = text.angle, cex = y.axis.magnific)**

**}else{**

**stop("\n\nARGUMENT y.side CAN ONLY BE 2 OR 4\n\n")**

**}**

**par(xpd = FALSE)**

**y.text <- (par("usr")[4] + (par("usr")[4] - par("usr")[3]) / (par("plt")[4] - par("plt")[3]) \* (1 - par("plt")[4]))**

**}else{**

**stop("\n\nPROBLEM WITH THE y.side (", y.side ,") OR y.log.scale (", y.log.scale,") ARGUMENTS\n\n")**

**}**

**}else{**

**y.text <- (par("usr")[4] + (par("usr")[4] - par("usr")[3]) / (par("plt")[4] - par("plt")[3]) \* (1 - par("plt")[4]))**

**}**

**par(xpd=NA)**

**text(x = x.mid.right.fig.region, y = y.text, corner.text, adj=c(1, 1.1), cex = magnific.corner.text)** *# text at the topright corner. Replace x.right.fig.region by x.text if text at the right edge of the plot region*

**if(just.label.add == TRUE & x.side == 0 & x.lab != ""){**

**text(x = x.mid.plot.region, y = y.mid.bottom.fig.region, x.lab, adj=c(0.5, 0.5), cex = x.label.magnific)** *# x label*

**}**

**if(just.label.add == TRUE & y.side == 0 & y.lab != ""){**

**text(x = y.mid.plot.region, y = x.mid.left.fig.region, y.lab, adj=c(0.5, 0.5), cex = y.label.magnific)** *# x label*

**}**

**par(xpd=FALSE)**

**if(par.reset == TRUE){**

**tempo.par <- fun\_open\_window(pdf.disp = FALSE, return.output = TRUE)**

**invisible(dev.off())** *# close the new window*

**if( ! is.null(custom.par)){**

**if( ! names(custom.par) %in% names(tempo.par$ini.par)){**

**tempo.cat <- paste0("\n\n================\n\nERROR IN ", function.name, ": custom.par ARGUMENT SHOULD HAVE THE NAMES OF THE COMPARTMENT LIST COMING FROM THE par() LIST\n\n================\n\n")**

**stop(tempo.cat)**

**}**

**par(custom.par)**

**text <- c(text, "\nGRAPH PARAMETERS SET TO VALUES DEFINED BY custom.par ARGUMENT\n")**

**}else{**

**par(tempo.par$ini.par)**

**text <- c(text, "\nGRAPH PARAMETERS RESET TO par() DEFAULT VALUES\n")**

**}**

**}**

**output <- list(x.mid.left.dev.region = x.mid.left.dev.region, x.left.dev.region = x.left.dev.region, x.mid.right.dev.region = x.mid.right.dev.region, x.right.dev.region = x.right.dev.region, x.mid.left.fig.region = x.mid.left.fig.region, x.left.fig.region = x.left.fig.region, x.mid.right.fig.region = x.mid.right.fig.region, x.right.fig.region = x.right.fig.region, x.left.plot.region = x.left.plot.region, x.right.plot.region = x.right.plot.region, x.mid.plot.region = x.mid.plot.region, y.mid.bottom.dev.region = y.mid.bottom.dev.region, y.bottom.dev.region = y.bottom.dev.region, y.mid.top.dev.region = y.mid.top.dev.region, y.top.dev.region = y.top.dev.region, y.mid.bottom.fig.region = y.mid.bottom.fig.region, y.bottom.fig.region = y.bottom.fig.region, y.mid.top.fig.region = y.mid.top.fig.region, y.top.fig.region = y.top.fig.region, y.top.plot.region = y.top.plot.region, y.bottom.plot.region = y.bottom.plot.region, y.mid.plot.region = y.mid.plot.region, text = text)**

**return(output)**

**}**

######## fun\_close\_specif\_window() #### Closing specific graphic windows

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

**fun\_close\_specif\_window <- function(kind = "pdf", return.text = FALSE){**

*# AIM*

*# close only specific graphic windows (devices)*

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

*# fun\_param\_check()*

*# ARGUMENTS:*

*# kind: vector, among c("windows", "quartz", "x11", "X11", "pdf", "bmp", "png", "tiff"), indicating the kind of graphic windows (devices) to close. BEWARE: either "windows", "quartz", "x11" or "X11" means that all the X11 GUI graphics devices will be closed, whatever the OS used*

*# return.text: print text regarding the kind parameter and the devices that were finally closed?*

*# RETURN*

*# text regarding the kind parameter and the devices that were finally closed*

*# EXAMPLES*

*# windows() ; windows() ; pdf() ; dev.list() ; fun\_close\_specif\_window(kind = c("pdf", "x11"), return.text = TRUE) ; dev.list()*

*# DEBUGGING*

*# kind = c("windows", "pdf") ; return.text = FALSE # for function debugging*

*# function name*

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

*# end function name*

*# required function checking*

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

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

**stop(tempo.cat)**

**}**

*# end required function checking*

*# argument checking*

**arg.check <- NULL** *# for function debbuging*

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

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

**tempo <- fun\_param\_check(data = kind, options = c("windows", "quartz", "x11", "X11", "pdf", "bmp", "png", "tiff"), fun.name = function.name) ; eval(ee)**

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

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

**stop()** *# nothing else because print = TRUE by default in fun\_param\_check()*

**}**

*# 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\_param\_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\_param\_check()*

*# end argument checking*

*# main code*

**text <- paste0("THE REQUIRED KIND OF GRAPHIC DEVICES TO CLOSE ARE ", paste(kind, collapse = " "))**

**if(Sys.info()["sysname"] == "Windows"){** *# Note that .Platform$OS.type() only says "unix" for macOS and Linux and "Windows" for Windows*

**if(any(kind %in% c("windows", "quartz", "x11", "X11"))){**

**tempo <- kind %in% c("windows", "quartz", "x11", "X11")**

**kind[tempo] <- "windows"** *# term are replaced by what is displayed when using a <- dev.list() ; names(a)*

**}**

**}else if(Sys.info()["sysname"] == "Linux"){**

**if(any(kind %in% c("windows", "quartz", "x11", "X11"))){**

**tempo.device <- suppressWarnings(try(X11(), silent = TRUE))[]** *# open a X11 window to try to recover the X11 system used*

**if( ! is.null(tempo.device)){**

**text <- paste0(text, "\nCANNOT CLOSE GUI GRAPHIC DEVICES AS REQUIRED BECAUSE THIS LINUX SYSTEM DOES NOT HAVE IT")**

**}else{**

**tempo <- kind %in% c("windows", "quartz", "x11", "X11")**

**kind[tempo] <- names(dev.list()[length(dev.list())])** *# term are replaced by what is displayed when using a <- dev.list() ; names(a)*

**invisible(dev.off())** *# close the X11 opened by tempo*

**}**

**}**

**}else{** *# for macOS*

**if(any(kind %in% c("windows", "quartz", "x11", "X11"))){**

**tempo <- kind %in% c("windows", "quartz", "x11", "X11")**

**kind[tempo] <- "quartz"** *# term are replaced by what is displayed when using a <- dev.list() ; names(a)*

**}**

**}**

**kind <- unique(kind)**

**if(length(dev.list()) != 0){**

**for(i in length(names(dev.list())):1){**

**if(names(dev.list())[i] %in% kind){**

**text <- paste0(text, "\n", names(dev.list())[i], " DEVICE NUMBER ", dev.list()[i], " HAS BEEN CLOSED")**

**invisible(dev.off(dev.list()[i]))**

**}**

**}**

**}**

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

**return(text)**

**}**

**}**

################ Standard graphics

######## fun\_empty\_graph() #### text to display for empty graphs

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

**fun\_empty\_graph <- function(text, title = NULL, text.size = 1){**

*# AIM*

*# display an empty plot with a text in the middle of the window (for instance to specify that no plot can be drawn)*

*# ARGUMENTS*

*# text: character string of the message to display*

*# text.size: numeric value of the text size*

*# REQUIRED PACKAGES*

*# none*

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

*# fun\_param\_check()*

*# RETURN*

*# an empty plot*

*# EXAMPLES*

*# fun\_empty\_graph(text = "NO GRAPH", title = "GRAPH1")*

*# DEBUGGING*

*# text = "NO GRAPH" ; title = "GRAPH1" ; text.size = 1*

*# function name*

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

*# end function name*

*# required function checking*

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

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

**stop(tempo.cat)**

**}**

*# end required function checking*

*# argument checking*

**arg.check <- NULL** *# for function debbuging*

**checked.arg.names <- NULL** *# for function debbuging*

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

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

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

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

**stop()** *# nothing else because print = TRUE by default in fun\_param\_check()*

**}**

*# 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\_param\_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\_param\_check()*

*# end argument checking*

*# main code*

**ini.par <- par(no.readonly = TRUE)** *# to recover the initial graphical parameters if required (reset). BEWARE: this command alone opens a pdf of GUI window if no window already opened. But here, protected with the code because always a tempo window opened*

**par(ann=FALSE, xaxt="n", yaxt="n", mar = rep(1, 4), bty = "n", xpd = NA)**

**plot(1, 1, type = "n")** *# no display with type = "n"*

**x.left.dev.region <- (par("usr")[1] - ((par("usr")[2] - par("usr")[1]) / (par("plt")[2] - par("plt")[1])) \* par("plt")[1] - ((par("usr")[2] - par("usr")[1]) / ((par("omd")[2] - par("omd")[1]) \* (par("plt")[2] - par("plt")[1]))) \* par("omd")[1])**

**y.top.dev.region <- (par("usr")[4] + ((par("usr")[4] - par("usr")[3]) / (par("plt")[4] - par("plt")[3])) \* (1 - par("plt")[4]) + ((par("usr")[4] - par("usr")[3]) / ((par("omd")[4] - par("omd")[3]) \* (par("plt")[4] - par("plt")[3]))) \* (1 - par("omd")[4]))**

**text(x = 1, y = 1, labels = text, cex = text.size)**

**text(x = x.left.dev.region, y = y.top.dev.region, labels = title, adj=c(0, 1), cex = text.size)**

**par(ini.par)**

**}**

################ gg graphics

######## fun\_gg\_palette() #### ggplot2 default color palette

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

**fun\_gg\_palette <- function(n){**

*# AIM*

*# provide colors used by ggplot2*

*# the interest is to use another single color that is not the red one used by default*

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

*# ARGUMENTS*

*# n: number of groups on the graph*

*# REQUIRED PACKAGES*

*# none*

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

*# fun\_param\_check()*

*# RETURN*

*# the vector of hexadecimal colors*

*# EXAMPLES*

*# fun\_gg\_palette(n = 2)*

*# plot(1, pch = 16, cex = 5, col = fun\_gg\_palette(n = 2)[2]) # second color of the two color ggplot2 palette*

*# DEBUGGING*

*# n = 0*

*# function name*

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

*# end function name*

*# required function checking*

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

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

**stop(tempo.cat)**

**}**

*# end required function checking*

*# argument checking*

**arg.check <- NULL** *# for function debbuging*

**checked.arg.names <- NULL** *# for function debbuging*

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

**tempo <- fun\_param\_check(data = n, class = "integer", length = 1, double.as.integer.allowed = TRUE, neg.values = FALSE, fun.name = function.name) ; eval(ee)**

**if(tempo$problem == FALSE & n == 0){**

**tempo.cat <- paste0("\n\n================\n\nERROR IN ", function.name, ": n ARGUMENT MUST BE A NON ZERO INTEGER. HERE IT IS: ", paste(n, collapse = " "), "\n\n================\n\n")**

**stop(tempo.cat)**

**}**

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

**stop()** *# nothing else because print = TRUE by default in fun\_param\_check()*

**}**

*# 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\_param\_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\_param\_check()*

*# end argument checking*

*# main code*

**hues = seq(15, 375, length = n + 1)**

**hcl(h = hues, l = 65, c = 100)[1:n]**

**}**

######## 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", xlim = NULL, ylim = NULL, extra.margin = 0.05, xlab = NULL, ylab = NULL, pt.size = 1, li.size = 0.5, alpha = 0.5, title = "", text.size = 12, return = FALSE, classic = FALSE, path.lib = NULL){**

*# AIM*

*# ggplot2 scatterplot with the possibility to overlay dots from up to 3 different data frames and lines from up to 3 different data frames (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*

*# 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 hline or vline geom*

*# 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) y argument must be NULL, (2) 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*

*# xlim: 2 numeric values for x-axis range. If NULL, range of x of all the data frames in data1*

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

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

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

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

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

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

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

*# title: character string of the graph title*

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

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

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

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

*# REQUIRED PACKAGES*

*# ggplot2*

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

*# fun\_param\_check()*

*# fun\_pack\_import()*

*# fun\_gg\_palette()*

*# RETURN*

*# a scatter plot*

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

*# data: the graphic info coordinates*

*# warnings: the warning messages*

*# EXAMPLES*

*# obs1 <- data.frame(a = 1:6, b = (1:6)^2, group = c("A", "A", "A", "B", "B", "B")) ; obs1 ; fun\_gg\_scatter(data1 = list(L1 = obs1), x = list(L1 = names(obs1)[1]), y = list(L1 = names(obs1)[2]), categ = list(L1 = names(obs1)[3]), legend.name = NULL, color = NULL, geom = list(L1 = "geom\_point"), xlim = NULL, ylim = NULL, extra.margin = 0.05, xlab = NULL, ylab = NULL, pt.size = 1, li.size = 0.5, alpha = 0.5, title = "GRAPH1", text.size = 12, return = FALSE, classic = FALSE)*

*# obs1 <- data.frame(a = 1:6, b = (1:6)^2, group = c("A", "A", "A", "B", "B", "B")) ; obs1 ; fun\_gg\_scatter(data1 = list(L1 = obs1), x = list(L1 = names(obs1)[1]), y = list(L1 = names(obs1)[2]), categ = NULL, legend.name = NULL, geom = list(L1 = "geom\_point"), xlim = NULL, ylim = NULL, extra.margin = 0.05, xlab = "test\_x", ylab = "test\_y", color = list(L1 = 5), pt.size = 2, li.size = 0.5, alpha = 1, title = "GRAPH1", text.size = 15, return = TRUE, classic = FALSE)*

*# obs1 <- data.frame(a = 1:6, b = (1:6)^2, group = c("A", "A", "A", "B", "B", "B")) ; obs1 ; fun\_gg\_scatter(data1 = list(L1 = obs1), x = list(L1 = names(obs1)[1]), y = list(L1 = names(obs1)[2]), categ = NULL, legend.name = NULL, geom = list(L1 = "geom\_path"), xlim = NULL, ylim = NULL, extra.margin = 0.05, xlab = "test\_x", ylab = "test\_y", color = list(L1 = 5), pt.size = 2, li.size = 0.5, alpha = 1, title = "GRAPH1", text.size = 15, return = TRUE, classic = FALSE)*

*# 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"))) ; data1 ; fun\_gg\_scatter(data1 = data1, x = list(L1 = names(data1$L1)[1], L2 = names(data1$L2)[1]), y = list(L1 = names(data1$L1)[2], L2 = names(data1$L2)[2]), categ = list(L1 = names(data1$L1)[3], L2 = names(data1$L2)[3]), legend.name = list(L1 = "GROUP1", L2 = "GROUP2"), color = list(L1 = fun\_gg\_palette(4)[1:2], L2 = fun\_gg\_palette(4)[3:4]), geom = list(L1 = "geom\_point", L2 = "geom\_point"), xlim = NULL, ylim = NULL, extra.margin = 0.05, xlab = NULL, ylab = NULL, pt.size = 2, li.size = 0.5, alpha = 0.5, title = "GRAPH1", text.size = 12, return = TRUE, classic = FALSE, path.lib = 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("A2", "A2", "A3", "A3", "B1", "B1"))) ; data1 ; fun\_gg\_scatter(data1 = data1, x = list(L1 = names(data1$L1)[1], L2 = names(data1$L2)[1], L3 = names(data1$L3)[1]), y = list(L1 = names(data1$L1)[2], L2 = names(data1$L2)[2], L3 = names(data1$L3)[2]), categ = list(L1 = names(data1$L1)[3], L2 = names(data1$L2)[3], L3 = names(data1$L3)[3]), legend.name = NULL, color = list(L1 = fun\_gg\_palette(7)[1:2], L2 = fun\_gg\_palette(7)[3:4], L3 = fun\_gg\_palette(7)[5:7]), geom = list(L1 = "geom\_point", L2 = "geom\_point", L3 = "geom\_path"), xlim = NULL, ylim = NULL, extra.margin = 0.05, xlab = NULL, ylab = NULL, pt.size = 4, li.size = 0.5, alpha = 0.5, title = "GRAPH1", text.size = 12, return = TRUE, classic = FALSE, path.lib = 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("A2", "A2", "A3", "A3", "B1", "B1"))) ; data1 ; fun\_gg\_scatter(data1 = data1, x = list(L1 = names(data1$L1)[1], L2 = names(data1$L2)[1], L3 = names(data1$L3)[1]), y = list(L1 = names(data1$L1)[2], L2 = names(data1$L2)[2], L3 = names(data1$L3)[2]), categ = list(L1 = names(data1$L1)[3], L2 = names(data1$L2)[3], NULL), legend.name = NULL, color = list(L1 = fun\_gg\_palette(7)[1:2], L2 = fun\_gg\_palette(7)[3:4], L3 = NULL), geom = list(L1 = "geom\_point", L2 = "geom\_point", L3 = "geom\_path"), xlim = NULL, ylim = NULL, extra.margin = 0.05, xlab = NULL, ylab = NULL, pt.size = 4, li.size = 0.5, alpha = 0.5, title = "GRAPH1", text.size = 12, return = TRUE, classic = FALSE, path.lib = 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 ; fun\_gg\_scatter(data1 = data1, x = list(L1 = names(data1$L1)[1], L2 = names(data1$L2)[1], L3 = names(data1$L3)[1]), y = list(L1 = names(data1$L1)[2], L2 = names(data1$L2)[2], L3 = NULL), categ = list(L1 = names(data1$L1)[3], L2 = names(data1$L2)[3], L3 = names(data1$L3)[3]), legend.name = NULL, color = list(L1 = "red", L2 = "blue", L3 = "green"), geom = list(L1 = "geom\_point", L2 = "geom\_point", L3 = "geom\_vline"), xlim = NULL, ylim = NULL, extra.margin = 0.05, xlab = NULL, ylab = NULL, pt.size = 4, li.size = 0.5, alpha = 0.5, title = "GRAPH1", text.size = 12, return = TRUE, classic = FALSE, path.lib = 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 ; data1 ; fun\_gg\_scatter(data1 = data1, x = list(L1 = names(data1$L1)[1], L2 = names(data1$L2)[1], L3 = names(data1$L3)[1]), y = list(L1 = names(data1$L1)[2], L2 = names(data1$L2)[2], L3 = NULL), categ = list(L1 = names(data1$L1)[3], L2 = names(data1$L2)[3], L3 = names(data1$L3)[3]), legend.name = NULL, color = list(L1 = "red", L2 = "blue", L3 = "green"), geom = list(L1 = "geom\_point", L2 = "geom\_point", L3 = "geom\_vline"), xlim = NULL, ylim = NULL, extra.margin = 0.05, xlab = NULL, ylab = NULL, pt.size = 4, li.size = 0.5, alpha = 0.5, title = "GRAPH1", text.size = 12, return = TRUE, classic = FALSE, path.lib = 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("A2", "A2", "A3", "A3", "B1", "B1"))) ; data1 ; fun\_gg\_scatter(data1 = data1, x = list(L1 = names(data1$L1)[1], L2 = names(data1$L2)[1], L3 = names(data1$L3)[1]), y = list(L1 = names(data1$L1)[2], L2 = names(data1$L2)[2], L3 = names(data1$L3)[2]), categ = NULL, legend.name = list(L1 = "A", L2 = "B", L3 = "C"), color = list(L1 = "black", L2 = 2, L3 = "purple"), geom = list(L1 = "geom\_point", L2 = "geom\_point", L3 = "geom\_point"), xlim = NULL, ylim = NULL, extra.margin = 0.05, xlab = NULL, ylab = NULL, pt.size = 1, li.size = 0.5, alpha = 1, title = "GRAPH1", text.size = 20, return = TRUE, classic = TRUE, path.lib = NULL)*

*# data1 <- list(L1 = data.frame(a = 1:6, b = (1:6)^2, group = c("A1", "A2", "A3", "B1", "B2", "B3"))) ; data1$L1$a[2:3] <- NA ; x = list(L1 = names(data1$L1)[1]) ; y = list(L1 = NULL) ; categ = list(L1 = names(data1$L1)[3]) ; data1 ; fun\_gg\_scatter(data1 = data1, x = list(L1 = names(data1$L1)[1]), y = list(L1 = NULL), categ = list(L1 = names(data1$L1)[3]), legend.name = list(L1 = "VALUE"), color = list(L1 = "red"), geom = list(L1 = "geom\_hline"), xlim = NULL, ylim = NULL, extra.margin = 0.05, xlab = NULL, ylab = NULL, pt.size = 1, li.size = 0.5, alpha = 0.5, title = "GRAPH1", text.size = 12, return = TRUE, classic = TRUE, path.lib = NULL)*

*# DEBUGGING*

*# data1 <- data.frame(a = 1:6, b = (1:6)^2, group = c("A", "A", "A", "B", "B", "B")) ; x = names(data1)[1] ; y = names(data1)[2] ; categ = names(data1)[3] ; legend.name = NULL ; color = NULL ; geom = "geom\_point" ; xlim = NULL ; ylim = NULL ; extra.margin = 0.05 ; xlab = NULL ; ylab = NULL ; pt.size = 1 ; li.size = 0.5 ; alpha = 0.5 ; title = "GRAPH1" ; text.size = 12 ; return = FALSE ; classic = FALSE ; path.lib = NULL*

*# data1 <- list(L1 = data.frame(a = 1:6, b = (1:6)^2, group = c("A", "A", "A", "B", "B", "B"))) ; x = list(L1 = names(data1$L1)[1]) ; y = list(L1 = names(data1$L1)[2]) ; categ = list(L1 = names(data1$L1)[3]) ; legend.name = list(L1 = "VALUE") ; color = NULL ; geom = list(L1 = "geom\_point") ; xlim = NULL ; ylim = NULL ; extra.margin = 0.05 ; xlab = NULL ; ylab = NULL ; pt.size = 1 ; li.size = 0.5 ; alpha = 0.5 ; title = "GRAPH1" ; text.size = 12 ; return = FALSE ; classic = FALSE ; path.lib = 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"))) ; x = list(L1 = names(data1$L1)[1], L2 = names(data1$L2)[1]) ; y = list(L1 = names(data1$L1)[2], L2 = names(data1$L2)[2]) ; categ = list(L1 = names(data1$L1)[3], L2 = names(data1$L2)[3]) ; legend.name = list(L1 = "GROUP1", L2 = "GROUP2") ; color = NULL ; geom = list(L1 = "geom\_point", L2 = "geom\_path") ; xlim = NULL ; ylim = NULL ; extra.margin = 0.05 ; xlab = NULL ; ylab = NULL ; pt.size = 1 ; li.size = 0.5 ; alpha = 0.5 ; title = "GRAPH1" ; text.size = 12 ; return = FALSE ; classic = FALSE ; path.lib = NULL*

*# data1 <- list(L1 = data.frame(a = 1:6, b = (1:6)^2, group1 = c("A", "A", "A", "B", "B", "B")), L2 = data.frame(a = (1:6)\*2, b = ((1:6)^2)\*2, group2 = c("A1", "A1", "A1", "B1", "B1", "B1"))) ; x = list(L1 = names(data1$L1)[1], L2 = names(data1$L2)[1]) ; y = list(L1 = names(data1$L1)[2], L2 = names(data1$L2)[2]) ; categ = list(L1 = names(data1$L1)[3], L2 = names(data1$L2)[3]) ; legend.name = list(L1 = "GROUP1", L2 = "GROUP2") ; color = list(L1 = 1:2, L2 = 3:4) ; geom = list(L1 = "geom\_point", L2 = "geom\_line") ; xlim = NULL ; ylim = NULL ; extra.margin = 0.05 ; xlab = NULL ; ylab = NULL ; pt.size = 2 ; li.size = 0.5 ; alpha = 0.5 ; title = "GRAPH1" ; text.size = 12 ; return = TRUE ; classic = FALSE ; path.lib = NULL*

*# data1 <- list(L1 = data.frame(a = 1:6, b = (1:6)^2, group = c("A", "A", "A", "B", "B", "B"))) ; x = list(L1 = names(data1$L1)[1]) ; y = list(L1 = names(data1$L1)[2]) ; categ = NULL ; legend.name = NULL ; color = list(L1 = 5) ; geom = list(L1 = "geom\_point") ; xlim = NULL ; ylim = NULL ; extra.margin = 0.05 ; xlab = "x test" ; ylab = "y test" ; pt.size = 2 ; li.size = 0.5 ; alpha = 1 ; title = "GRAPH1" ; text.size = 15 ; return = TRUE ; classic = FALSE ; path.lib = NULL*

*# data1 <- list(L1 = data.frame(a = 1:6, b = (1:6)^2, group = c("A1", "A2", "A3", "B1", "B2", "B3"))) ; x = list(L1 = names(data1$L1)[1]) ; y = list(L1 = NULL) ; categ = list(L1 = names(data1$L1)[3]) ; legend.name = list(L1 = "VALUE") ; color = list(L1 = "red") ; geom = list(L1 = "geom\_hline") ; xlim = NULL ; ylim = NULL ; extra.margin = 0.05 ; xlab = NULL ; ylab = NULL ; pt.size = 1 ; li.size = 0.5 ; alpha = 0.5 ; title = "GRAPH1" ; text.size = 12 ; return = TRUE ; classic = TRUE ; path.lib = NULL*

*# data1 <- list(L1 = data.frame(a = 1:6, b = (1:6)^2, group = c("A1", "A2", "A3", "B1", "B2", "B3"))) ; data1$L1$a[2:3] <- NA ; x = list(L1 = names(data1$L1)[1]) ; y = list(L1 = NULL) ; categ = list(L1 = names(data1$L1)[3]) ; legend.name = list(L1 = "VALUE") ; color = list(L1 = "red") ; geom = list(L1 = "geom\_hline") ; xlim = NULL ; ylim = NULL ; extra.margin = 0.05 ; xlab = NULL ; ylab = NULL ; pt.size = 1 ; li.size = 0.5 ; alpha = 0.5 ; title = "GRAPH1" ; text.size = 12 ; return = TRUE ; classic = TRUE ; path.lib = 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 = names(data1$L3)[1]) ; y = list(L1 = names(data1$L1)[2], L2 = names(data1$L2)[2], L3 = NULL) ; categ = list(L1 = names(data1$L1)[3], L2 = names(data1$L2)[3], L3 = names(data1$L3)[3]) ; legend.name = NULL ; color = list(L1 = "red", L2 = "blue", L3 = "green") ; geom = list(L1 = "geom\_point", L2 = "geom\_point", L3 = "geom\_vline") ; xlim = NULL ; ylim = NULL ; extra.margin = 0.05 ; xlab = NULL ; ylab = NULL ; pt.size = 4 ; li.size = 0.5 ; alpha = 0.5 ; title = "GRAPH1" ; text.size = 12 ; return = TRUE ; classic = FALSE ; path.lib = NULL*

*# function name*

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

*# end function name*

*# required function checking*

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

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

**stop(tempo.cat)**

**}**

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

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

**stop(tempo.cat)**

**}**

**if(length(find("fun\_gg\_palette", mode = "function")) == 0){**

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

**stop(tempo.cat)**

**}**

*# end required function checking*

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

**warning <- NULL**

**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)**

**}**

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

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

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

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

**}**

**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)**

**}**

**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)**

**}**

**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)**

**}**

**}**

**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)**

**}**

**}**

**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)**

**}**

**}**

**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)**

**}**

**}**

*# 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)**

**}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)**

**}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)**

**}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)**

**}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)**

**}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)**

**}else{**

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

**}**

**}**

*# 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*

*# 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)**

**}**

**tempo.check.color <- c(tempo.check.color, fun\_param\_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** *# for function debbuging*

**checked.arg.names <- NULL** *# for function debbuging*

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

**compart.null.color <- 0** *# will be used to attribute a color when color is non NULL but a compatment 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\_param\_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)**

**tempo <- fun\_param\_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)**

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

**tempo <- fun\_param\_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(y[[i1]])){**

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

**tempo.cat <- paste0("\n\n================\n\nERROR 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\_hline\" OR \"geom\_vline\"\nHERE geom ARGUMENT IS: ", paste(geom[[i1]], collapse = " "), "\n\n================\n\n")**

**stop(tempo.cat)**

**}else{**

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

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

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

**tempo.warning <- paste0("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")**

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

**}**

**}else{**

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

**tempo.cat <- paste0("\n\n================\n\nERROR 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 NON NULL IF ", ifelse(length(geom) == 1, "y", paste0("geom NUMBER ", i1)), " ARGUMENT IS \"geom\_hline\" OR \"geom\_vline\"\nHERE geom ARGUMENT IS: ", paste(geom[[i1]], collapse = " "), "\n\n================\n\n")**

**stop(tempo.cat)**

**}**

**tempo <- fun\_param\_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("\n\n================\n\nERROR 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)), "\n\n================\n\n")**

**stop(tempo.cat)**

**}**

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

**tempo.cat <- paste0("\n\n================\n\nERROR 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)), "\n\n================\n\n")**

**stop(tempo.cat)**

**}**

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

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

**tempo.removed.row.nb <- unlist(lapply(lapply(c(data1[[i1]][c(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, ]**

**tempo.warning <- paste0("NA DETECTED IN COLUMN ", ifelse(length(x) == 1, "x", paste0("x NUMBER ", i1)), if(y[[i1]] == "fake\_y"){""}else{paste0(" AND ", 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)")**

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

**}**

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

**tempo <- fun\_param\_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 = FALSE, fun.name = function.name) ; eval(ee)**

**tempo <- fun\_param\_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\_param\_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("\n\n================\n\nERROR 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)), "\n\n================\n\n")**

**stop(tempo.cat)**

**}**

*# 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, ]**

**tempo.warning <- paste0("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")**

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

**}**

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

**tempo1 <- fun\_param\_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\_param\_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("\n\n================\n\nERROR 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\n\n================\n\n")**

**stop(tempo.cat)**

**}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*

**tempo.warning <- paste0("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")**

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

**}**

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

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

**tempo.cat <- paste0("\n\n================\n\nERROR 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)\n\n================\n\n")**

**stop(tempo.cat)**

**}**

**}**

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

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

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

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

**tempo.warning <- paste0("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")**

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

**}**

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

**tempo <- fun\_param\_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*

**tempo.warning <- paste0("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")**

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

**}**

**tempo1 <- fun\_param\_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\_param\_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("\n\n================\n\nERROR IN ", function.name, ": ", ifelse(length(color) == 1, "color", paste0("color NUMBER ", i1)), " MUST BE A FACTOR OR CHARACTER VECTOR OR INTEGER VECTOR\n\n================\n\n")** *# integer possible because dealt above*

**stop(tempo.cat)**

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

**tempo.cat <- paste0("\n\n================\n\nERROR 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 = " "), "\n\n================\n\n")**

**stop(tempo.cat)**

**}**

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

**tempo.warning <- paste0("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")**

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

**}**

*# end check the nature of color*

*# check the length of color*

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

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

**stop(tempo.cat)**

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

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

**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*

**tempo.warning <- paste0("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 = " "))**

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

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

**tempo.cat <- paste0("\n\n================\n\nERROR 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"), "\n\n================\n\n")**

**stop(tempo.cat)**

**}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]]])))*

**tempo.warning <- paste0(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 = " "))**

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

**}**

**}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]]])))**

**tempo.warning <- paste0("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 = " "))**

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

**}else{**

**tempo.cat <- paste0("\n\n================\n\nERROR 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]]])), "\n\n================\n\n")**

**stop(tempo.cat)**

**}**

**}**

**}**

**}**

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

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

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

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

**}**

**}**

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

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

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

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

**}**

**}**

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

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

**stop(tempo.cat)**

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

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

**stop(tempo.cat)**

**}**

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

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

**}**

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

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

**}**

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

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

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

**}**

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

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

**}**

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

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

**tempo <- fun\_param\_check(data = alpha, prop = TRUE, length = 1, fun.name = function.name) ; eval(ee)**

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

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

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

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

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

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

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

**cat(paste0("\n\n============\n\nERROR IN ", function.name, ": \nDIRECTORY PATH INDICATED IN THE path.lib PARAMETER DOES NOT EXISTS: ", path.lib, "\n\n============\n\n"))**

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

**}**

**}**

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

**stop()** *# nothing else because print = TRUE by default in fun\_param\_check()*

**}**

*# 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\_param\_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\_param\_check()*

*# end argument checking*

*# package checking*

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

*# end package checking*

*# main code*

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

**for(i1 in 1:length(data1)){** *# for geom\_hline, x put into y, and x <- NA before xlim and ylim*

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

**data1[[i1]][, y[[i1]]] <- data1[[i1]][, x[[i1]]]**

**# data1[[i1]][, x[[i1]]] <- NA**

**}**

**}**

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

**tempo.x.range <- suppressWarnings(range(unlist(mapply(FUN = "[[", data1, x, SIMPLIFY = FALSE)), na.rm = TRUE))**

**tempo.y.range <- suppressWarnings(range(unlist(mapply(FUN = "[[", data1, y, SIMPLIFY = FALSE)), na.rm = TRUE))**

**if(suppressWarnings(all(tempo.x.range %in% c(Inf, -Inf))) | suppressWarnings(all(tempo.y.range %in% c(Inf, -Inf)))){**

**tempo.cat <- paste0("\n\n================\n\nERROR IN ", function.name, " geom\_hline AND geom\_vline CONVERSION: ", ifelse(length(x) == 1, "x", paste0("x NUMBER ", i1)), " AND ", ifelse(length(y) == 1, "y", paste0("y NUMBER ", i1)), " ARGUMENTS ARE NA ONLY\n\n================\n\n")**

**stop(tempo.cat)**

**}**

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

**# if(suppressWarnings(all(tempo.x.range %in% c(Inf, -Inf)))){**

**# xlim <- tempo.y.range** *# because of the switch for geom\_hline*

**# }else{**

**xlim <- tempo.x.range**

**# }**

**}**

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

**# if(suppressWarnings(all(tempo.y.range %in% c(Inf, -Inf)))){**

**# ylim <- tempo.x.range** *# because of the switch for geom\_hline*

**# }else{**

**ylim <- tempo.y.range**

**# }**

**}**

**if( ! is.null(extra.margin)){**

**xlim <- sort(xlim)**

**xlim[1] <- xlim[1] - abs(xlim[2] - xlim[1]) \* extra.margin**

**xlim[2] <- xlim[2] + abs(xlim[2] - xlim[1]) \* extra.margin**

**ylim <- sort(ylim)**

**ylim[1] <- ylim[1] - abs(ylim[2] - ylim[1]) \* extra.margin**

**ylim[2] <- ylim[2] + abs(ylim[2] - ylim[1]) \* extra.margin**

**}**

*# 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 = "")**

**}**

**tempo.warning <- paste0("NULL categ ARGUMENT -> FAKE COLUMN ADDED TO EACH DATA FRAME IN data1, NAMED \"fake\_categ\" AND FILLED WITH \"\"")**

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

**}**

*# 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)])})**

**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]]**

**tempo.warning <- paste0("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 = " "))**

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

**}**

**}**

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

*# last check*

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

**if(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)**

**}**

**}**

*# 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: CODE INCONSISTENCY\n\n============\n\n"))**

**stop(tempo.cat)**

**}**

**tempo.data.frame[, categ[[i1]]] <- paste0("Line\_", i3)**

**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: ", 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)**

**}**

**}**

**}**

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

**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))**

**assign(paste0(tempo.gg.name, tempo.gg.count <- tempo.gg.count + 1), ggplot2::scale\_x\_continuous(expand = c(0, 0), limits = NA))**

**assign(paste0(tempo.gg.name, tempo.gg.count <- tempo.gg.count + 1), ggplot2::scale\_y\_continuous(expand = c(0, 0), limits = NA))**

**assign(paste0(tempo.gg.name, tempo.gg.count <- tempo.gg.count + 1), ggplot2::coord\_cartesian(xlim = xlim, ylim = ylim))**

**if(classic == TRUE){**

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

**}else{**

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

**}**

*# end no need loop part*

**point.count <- 0**

**line.count <- 0**

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

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

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

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

**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), ggplot2::geom\_point(data = tempo.data.frame, mapping = ggplot2::aes\_string(x = x[[i1]], y = y[[i1]], fill = categ[[i1]]), size = pt.size, color = color[[i1]][i5], alpha = alpha, show.legend = TRUE))** *# beware: a single color allowed for color argumant outside aesthetic, hence the loop*

**}**

**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 = color[[i1]], guide = ggplot2::guide\_legend(override.aes = list(colour = color[[i1]], linetype = 0))))** *# values are the values of fill*

**}**

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

**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), ggplot2::geom\_point(data = tempo.data.frame, mapping = ggplot2::aes\_string(x = x[[i1]], y = y[[i1]], shape = categ[[i1]]), size = pt.size, color = color[[i1]][i5], alpha = alpha, show.legend = TRUE))** *# beware: a single color allowed for color argumant outside aesthetic, hence the loop*

**}**

**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){**

**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), ggplot2::geom\_point(data = tempo.data.frame, mapping = ggplot2::aes\_string(x = x[[i1]], y = y[[i1]], stroke = categ[[i1]]), size = pt.size, color = color[[i1]][i5], alpha = alpha, show.legend = TRUE))** *# beware: a single color allowed for color argumant outside aesthetic, hence the loop*

**}**

**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){**

**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), get(geom[[i1]])(data = tempo.data.frame, mapping = ggplot2::aes\_string(x = x[[i1]], y = y[[i1]], linetype = categ[[i1]]), color = color[[i1]][i5], size = li.size, lineend = "round", show.legend = TRUE))** *# beware: a single color allowed for color argumant outside aesthetic, hence the loop*

**}**

**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*

**}**

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

**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), get(geom[[i1]])(data = tempo.data.frame, mapping = ggplot2::aes\_string(x = x[[i1]], y = y[[i1]], alpha = categ[[i1]]), color = color[[i1]][i5], size = li.size, lineend = "round", show.legend = TRUE))** *# beware: a single color allowed for color argumant outside aesthetic, hence the loop*

**}**

**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(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*

**}**

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

**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), get(geom[[i1]])(data = tempo.data.frame, mapping = ggplot2::aes\_string(x = x[[i1]], y = y[[i1]], size = categ[[i1]]), color = color[[i1]][i5], size = li.size, lineend = "round", show.legend = TRUE))** *# beware: a single color allowed for color argumant outside aesthetic, hence the loop*

**}**

**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(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*

**}**

**}**

**}**

*# no need loop part*

*# end no need loop part*

**suppressWarnings(print(eval(parse(text = paste(paste0(tempo.gg.name, 1:tempo.gg.count), collapse = " + ")))))**

**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]], ]**

**}**

**}**

**}**

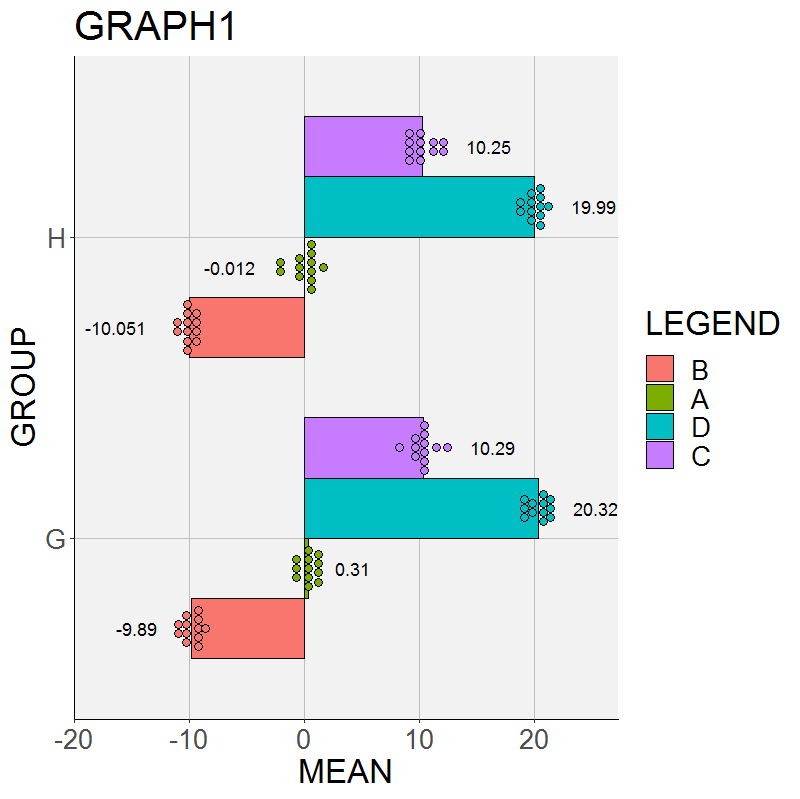
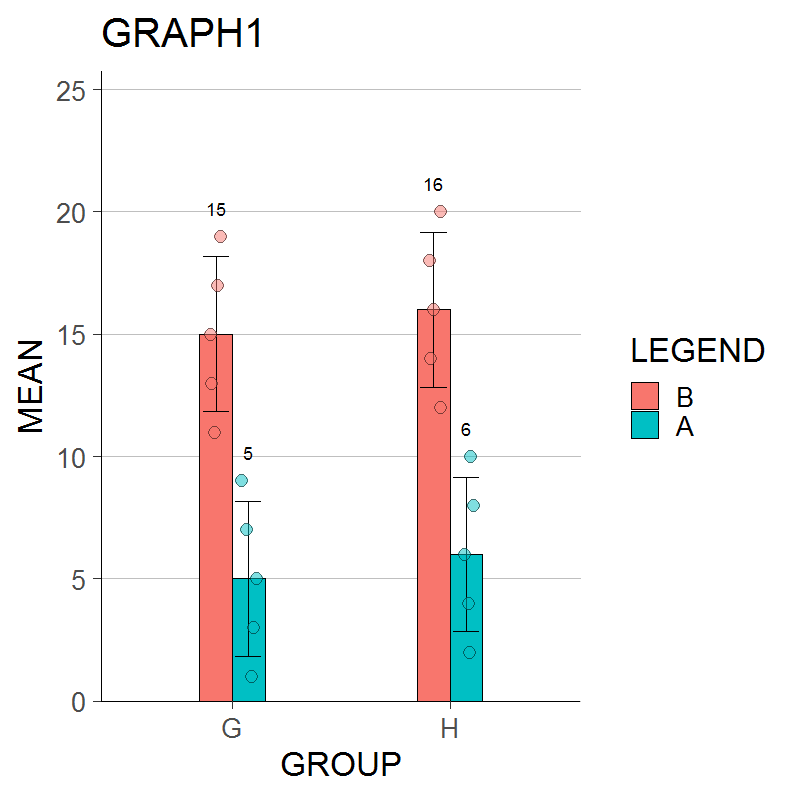
**output <- list(data = output$data, removed.row.nb = removed.row.nb, removed.rows = removed.rows, warnings = paste0("\n", warning, "\n\n"))**

**return(output)**

**}**

**}**

######## fun\_gg\_bar\_mean() #### ggplot2 mean barplot + overlaid dots if required



**# verif all coord objects, and output of aggregate anf merge to forbide some column names (like x.y) or pass them to upper case + warning**

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

**fun\_gg\_bar\_mean <- function(data1, y, categ, categ.class.order = NULL, categ.legend.name = NULL, categ.color = NULL, bar.width = 0.5, 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.break.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, classic = FALSE, grid = FALSE, return = FALSE, path.lib = 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\_mean() 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\_mean() 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, 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)*

*# 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 stroke*

*# 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. Log10 scale for the y-axis? 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.break.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)*

*# 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?*

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

*# REQUIRED PACKAGES*

*# ggplot2*

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

*# fun\_param\_check()*

*# fun\_pack\_import()*

*# fun\_gg\_palette()*

*# fun\_round()*

*# fun\_2D\_comp()*

*# RETURN*

*# a barplot*

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

*# stat: the graphic statistics*

*# removed.row.nb: which rows have been removed due to NA detection in y and categ columns (NULL if no row removed)*

*# removed.rows: removed rows containing NA (NULL if no row removed)*

*# data: the graphic info coordinates*

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

*# EXAMPLES*

*# nice representation (1)*

*# obs1 <- data.frame(a = 1:20, group1 = rep(c("G", "H"), times = 10), group2 = rep(c("A", "B"), each = 10)) ; fun\_gg\_bar\_mean(data1 = obs1, y = "a", categ = c("group1", "group2"), categ.class.order = list(NULL, c("B", "A")), categ.legend.name = "LEGEND", categ.color = NULL, dot.color = "same", error.disp = "SD", bar.width = 0.3, error.whisker.width = 0.8, dot.jitter = 0.5, ylim = c(10, 25), y.include.zero = TRUE, xlab = "GROUP", ylab = "MEAN", dot.size = 3.5, dot.border.size = 0.2, dot.alpha = 0.5, stat.disp = "above", stat.size = 4, title = "GRAPH1", text.size = 20, return = TRUE, y.break.nb = NULL, classic = TRUE, grid = TRUE)*

*# nice representation (2)*

*# set.seed(1) ; obs1 <- data.frame(a = c(rnorm(24, 0), rnorm(24, -10), rnorm(24, 10), rnorm(24, 20)), group1 = rep(c("G", "H"), times = 48), group2 = rep(c("A", "B", "C", "D"), each = 24)) ; set.seed(NULL) ; fun\_gg\_bar\_mean(data1 = obs1, y = "a", categ = c("group1", "group2"), categ.class.order = list(NULL, c("B", "A", "D", "C")), categ.legend.name = "LEGEND", categ.color = NULL, dot.color = "same", bar.width = 0.8, dot.tidy = TRUE, dot.bin.nb = 60, ylim= c(-20, 25), vertical = FALSE, xlab = "GROUP", ylab = "MEAN", dot.size = 3.5, dot.border.size = 0.2, dot.alpha = 1, stat.disp = "above", stat.size = 4, stat.dist = 1, title = "GRAPH1", text.size = 20, return = TRUE, y.break.nb = NULL, classic = FALSE)*

*# separate bars, modification of bar color (1) (a single value)*

*# obs1 <- data.frame(a = 1:20, group1 = rep(c("G", "H"), times = 10)) ; fun\_gg\_bar\_mean(data1 = obs1, y = "a", categ = "group1", categ.color = "white")*

*# separate bars, modification of bar color (2) (one value par class of categ2)*

*# obs1 <- data.frame(a = 1:20, group1 = rep(c("G", "H"), times = 10)) ; fun\_gg\_bar\_mean(data1 = obs1, y = "a", categ = "group1", categ.color = c("red", "blue"))*

*# separate bars, modification of bar color (3) (one value per line of obs1, with respect of the correspondence between categ2 and bar.color columns)*

*# obs1 <- data.frame(a = 1:20, group1 = rep(c("G", "H"), times = 10), bar.color = rep(c("brown", "orange"), time = 10)) ; obs1 ; fun\_gg\_bar\_mean(data1 = obs1, y = "a", categ = "group1", categ.color = obs1$bar.color)*

*# separate bars, modification of dot color (1) (same dot color as the corresponding bar)*

*# obs1 <- data.frame(a = 1:20, group1 = rep(c("G", "H"), times = 10)) ; fun\_gg\_bar\_mean(data1 = obs1, y = "a", categ = "group1", dot.color = "same")*

*# separate bars, modification of dot color (2) (single color for all the dots)*

*# obs1 <- data.frame(a = 1:20, group1 = rep(c("G", "H"), times = 10)) ; fun\_gg\_bar\_mean(data1 = obs1, y = "a", categ = "group1", dot.color = "green")*

*# separate bars, modification of dot color (3) (one value par class of categ2)*

*# obs1 <- data.frame(a = 1:20, group1 = rep(c("G", "H"), times = 10)) ; fun\_gg\_bar\_mean(data1 = obs1, y = "a", categ = "group1", dot.color = c("green", "brown"))*

*# separate bars, modification of dot color 4 (any color for each dot)*

*# obs1 <- data.frame(a = 1:10, group1 = rep(c("G", "H"), times = 5)) ; fun\_gg\_bar\_mean(data1 = obs1, y = "a", categ = "group1", dot.color = hsv(h = (1:nrow(obs1)) / nrow(obs1)))*

*# grouped bars, default arguments*

*# obs1 <- data.frame(a = 1:20, group1 = rep(c("G", "H"), times = 10), group2 = rep(c("A", "B"), each = 10)) ; fun\_gg\_bar\_mean(data1 = obs1, y = "a", categ = c("group1", "group2"))*

*# more grouped bars*

*# obs1 <- data.frame(a = 1:24, group1 = rep(c("G", "H"), times = 12), group2 = rep(c("A", "B", "C", "D"), each = 6)) ; fun\_gg\_bar\_mean(data1 = obs1, y = "a", categ = c("group1", "group2"), return = TRUE)*

*# grouped bars, log scale. Beware, y column must be log, otherwise incoherent scale*

*# obs1 <- data.frame(a = log10((1:20) \* 100), group1 = rep(c("G", "H"), times = 10), group2 = rep(c("A", "B"), each = 10)) ; fun\_gg\_bar\_mean(data1 = obs1, y = "a", categ = c("group1", "group2"), ylog = TRUE)*

*# grouped bars, no dots (y.include.zero = TRUE to see the lowest bar)*

*# obs1 <- data.frame(a = 1:20, group1 = rep(c("G", "H"), times = 10), group2 = rep(c("A", "B"), each = 10)) ; fun\_gg\_bar\_mean(data1 = obs1, y = "a", categ = c("group1", "group2"), dot.color = NULL, y.include.zero = TRUE)*

*# grouped bars, modification of bar color (1) (a single value)*

*# obs1 <- data.frame(a = 1:20, group1 = rep(c("G", "H"), times = 10), group2 = rep(c("A", "B"), each = 10)) ; fun\_gg\_bar\_mean(data1 = obs1, y = "a", categ = c("group1", "group2"), categ.color = "white")*

*# grouped bars, modification of bar color (2) (one value par class of categ2)*

*# obs1 <- data.frame(a = 1:20, group1 = rep(c("G", "H"), times = 10), group2 = rep(c("A", "B"), each = 10)) ; fun\_gg\_bar\_mean(data1 = obs1, y = "a", categ = c("group1", "group2"), categ.color = c("red", "blue"))*

*# grouped bars, modification of bar color (3) (one value per line of obs1, with respect of the correspondence between categ2 and bar.color columns)*

*# obs1 <- data.frame(a = 1:20, group1 = rep(c("G", "H"), times = 10), group2 = rep(c("A", "B"), each = 10), bar.color = rep(c("brown", "orange"), each = 10)) ; obs1 ; fun\_gg\_bar\_mean(data1 = obs1, y = "a", categ = c("group1", "group2"), categ.color = obs1$bar.color)*

*# grouped bars, modification of dot color (1) (same dot color as the corresponding bar)*

*# obs1 <- data.frame(a = 1:20, group1 = rep(c("G", "H"), times = 10), group2 = rep(c("A", "B"), each = 10)) ; fun\_gg\_bar\_mean(data1 = obs1, y = "a", categ = c("group1", "group2"), dot.color = "same")*

*# grouped bars, modification of dot color (2) (single color for all the dots)*

*# obs1 <- data.frame(a = 1:20, group1 = rep(c("G", "H"), times = 10), group2 = rep(c("A", "B"), each = 10)) ; fun\_gg\_bar\_mean(data1 = obs1, y = "a", categ = c("group1", "group2"), dot.color = "green")*

*# grouped bars, modification of dot color (3) (one value par class of categ2)*

*# obs1 <- data.frame(a = 1:20, group1 = rep(c("G", "H"), times = 10), group2 = rep(c("A", "B"), each = 10)) ; fun\_gg\_bar\_mean(data1 = obs1, y = "a", categ = c("group1", "group2"), dot.color = c("green", "brown"))*

*# grouped bars, modification of dot color (4) (any color for each dot)*

*# obs1 <- data.frame(a = 1:10, group1 = rep(c("G", "H"), times = 5), group2 = rep(c("A", "B"), each = 5)) ; fun\_gg\_bar\_mean(data1 = obs1, y = "a", categ = c("group1", "group2"), dot.color = hsv(h = (1:nrow(obs1)) / nrow(obs1)))*

*# width example. With bar.width = 0.25, three times more space between single bars than the bar width*

*# obs1 <- data.frame(a = 1:1000, group1 = rep(c("G", "H"), each = 500)) ; fun\_gg\_bar\_mean(data1 = obs1, y = "a", categ = "group1", dot.color = NULL, bar.width = 0.25, y.include.zero = TRUE)*

*# width example. With bar.width = 1, no space between single bars*

*# obs1 <- data.frame(a = 1:1000, group1 = rep(c("G", "H"), each = 500)) ; fun\_gg\_bar\_mean(data1 = obs1, y = "a", categ = "group1", dot.color = NULL, bar.width = 1, y.include.zero = TRUE)*

*# width example. With bar.width = 0.25, three times more space between sets of grouped bars than the set width*

*# obs1 <- data.frame(a = 1:1000, group1 = rep(c("G", "H"), times = 500), group2 = rep(LETTERS[1:5], each = 200)) ; fun\_gg\_bar\_mean(data1 = obs1, y = "a", categ = c("group1", "group2"), dot.color = NULL, bar.width = 0.25, y.include.zero = TRUE)*

*# width example. With bar.width = 0, no space between sets of grouped bars*

*# obs1 <- data.frame(a = 1:1000, group1 = rep(c("G", "H"), times = 500), group2 = rep(LETTERS[1:5], each = 200)) ; fun\_gg\_bar\_mean(data1 = obs1, y = "a", categ = c("group1", "group2"), dot.color = NULL, bar.width = 1, y.include.zero = TRUE)*

*# width example. With error.whisker.width = 1, whiskers have the width of the corresponding bar*

*# obs1 <- data.frame(a = 1:1000, group1 = rep(c("G", "H"), times = 500), group2 = rep(LETTERS[1:5], each = 200)) ; fun\_gg\_bar\_mean(data1 = obs1, y = "a", categ = c("group1", "group2"), dot.color = NULL, error.disp = "SD", error.whisker.width = 1)*

*# width example. No whiskers*

*# obs1 <- data.frame(a = 1:1000, group1 = rep(c("G", "H"), times = 500), group2 = rep(LETTERS[1:5], each = 200)) ; fun\_gg\_bar\_mean(data1 = obs1, y = "a", categ = c("group1", "group2"), dot.color = NULL, error.disp = "SD", error.whisker.width = 0)*

*# width example. With dot.jitter = 1, dispersion around the corresponding bar width*

*# obs1 <- data.frame(a = 1:1000, group1 = rep(c("G", "H"), times = 500), group2 = rep(LETTERS[1:5], each = 200)) ; fun\_gg\_bar\_mean(data1 = obs1, y = "a", categ = c("group1", "group2"), dot.color = "grey", dot.size = 3, dot.alpha = 1, dot.jitter = 1)*

*# width example. No dispersion*

*# obs1 <- data.frame(a = 1:1000, group1 = rep(c("G", "H"), times = 500), group2 = rep(LETTERS[1:5], each = 200)) ; fun\_gg\_bar\_mean(data1 = obs1, y = "a", categ = c("group1", "group2"), dot.color = "grey", dot.size = 3, dot.alpha = 1, dot.jitter = 0)*

*# tidy dot example*

*# obs1 <- data.frame(a = 1:1000, group1 = rep(c("G", "H"), times = 500), group2 = rep(LETTERS[1:5], each = 200)) ; fun\_gg\_bar\_mean(data1 = obs1, y = "a", categ = c("group1", "group2"), dot.color = "same", dot.tidy = TRUE, dot.bin.nb = 100)*

*# tidy dot example. Comparison with random spreading*

*# obs1 <- data.frame(a = 1:1000, group1 = rep(c("G", "H"), times = 500), group2 = rep(LETTERS[1:5], each = 200)) ; fun\_gg\_bar\_mean(data1 = obs1, y = "a", categ = c("group1", "group2"), dot.color = "same", dot.tidy = FALSE, dot.jitter = 1, dot.size = 2)*

*# orientation example. Beware: log scale automatically set to FALSE for horizontal display, because of a bug in ggplot2 (https://github.com/tidyverse/ggplot2/issues/881)*

*# obs1 <- data.frame(a = 1:1000, group1 = rep(c("G", "H"), times = 500), group2 = rep(LETTERS[1:5], each = 200)) ; fun\_gg\_bar\_mean(data1 = obs1, y = "a", categ = c("group1", "group2"), dot.color = "grey", dot.size = 3, dot.alpha = 1, dot.jitter = 1, vertical = FALSE)*

*# many arguments*

*# obs1 <- data.frame(a = 1:20, group1 = rep(c("G", "H"), times = 10), group2 = rep(c("A", "B"), each = 10)) ; fun\_gg\_bar\_mean(data1 = obs1, y = "a", categ = c("group1", "group2"), categ.class.order = list(NULL, c("B", "A")), categ.legend.name = "", categ.color = c("red", "blue"), bar.width = 0.25, error.disp = "SD", error.whisker.width = 0.8, dot.color = "grey", dot.jitter = 1, dot.size = 4, dot.border.size = 0, dot.alpha = 1, ylim = NULL, ylog = FALSE, y.break.nb = NULL, y.include.zero = FALSE, y.top.extra.margin = 0.05, y.bottom.extra.margin = 0, stat.disp = "above", stat.size = 4, xlab = "GROUP", ylab = "MEAN", vertical = TRUE, title = "GRAPH1", text.size = 14, classic = TRUE, grid = TRUE, return = TRUE, path.lib = NULL)*

*# DEBUGGING*

*# data1 <- data.frame(a = 1:20, group1 = rep(c("G", "H"), times = 10), group2 = rep(c("A", "B"), each = 10), bar.color = rep(c("brown", "orange"), each = 10)) ; data1[2:3, 1] <- NA ; data1[7:8, 2] <- NA ; y = names(data1)[1] ; categ = c(names(data1)[2], names(data1)[3]) ; categ.class.order = list(L1 = NULL, L2 = c("B", "A")) ; categ.legend.name = NULL ; categ.color = na.omit(data1)$bar.color ; bar.width = 0.5 ; error.disp = "SD" ; 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 = 1 ; ylim = NULL ; ylog = FALSE ; y.break.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 ; classic = FALSE ; grid = FALSE ; return = FALSE ; path.lib = NULL*

*# data1 <-data.frame(a = rep(1:20, 5), group1 = rep(c("G", "H"), times = 50), group2 = rep(LETTERS[1:5], each = 20)) ; y = names(data1)[1] ; categ = c(names(data1)[2], names(data1)[3]) ; categ.class.order = list(L1 = NULL, L2 = c("B", "A", "E", "D", "C")) ; categ.legend.name = NULL ; categ.color = NULL ; bar.width = 0.5 ; error.disp = "SD" ; error.whisker.width = 0.5 ; dot.color = "same" ; dot.tidy = TRUE ; dot.bin.nb = 30 ; dot.jitter = 0.25 ; dot.size = 3 ; dot.border.size = 0.5 ; dot.alpha = 1 ; ylim = NULL ; ylog = FALSE ; y.break.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 ; classic = FALSE ; grid = FALSE ; return = FALSE ; path.lib = NULL*

*# data1 <- data.frame(a = 1:20, group1 = rep(c("G", "H"), times = 10), group2 = rep(c("A", "B"), each = 10), bar.color = rep(c("brown", "orange"), each = 10)) ; data1[2:3, 1] <- NA ; data1[7:8, 2] <- NA ; y = names(data1)[1] ; categ = c(names(data1)[2], names(data1)[3]) ; categ.class.order = list(L1 = NULL, L2 = c("B", "A")) ; categ.legend.name = NULL ; categ.color = na.omit(data1)$bar.color ; bar.width = 0.5 ; error.disp = "SD" ; error.whisker.width = 0.5 ; dot.color = "same" ; dot.tidy = TRUE ; dot.bin.nb = 30 ; dot.jitter = 0.25 ; dot.size = 3 ; dot.border.size = 0.5 ; dot.alpha = 1 ; ylim = NULL ; ylog = FALSE ; y.break.nb = NULL ; y.include.zero = FALSE ; y.top.extra.margin = 0.05 ; y.bottom.extra.margin = 0 ; stat.disp = "above" ; stat.size = 4 ; stat.dist = 2 ; xlab = NULL ; ylab = NULL ; vertical = TRUE ; title = "" ; text.size = 12 ; classic = FALSE ; grid = FALSE ; return = FALSE ; path.lib = NULL*

*# set.seed(1) ; data1 <- data.frame(a = c(rnorm(25, 0), rnorm(25, -10), rnorm(25, 10), rnorm(25, 20)), group1 = rep(c("G", "H"), times = 50), group2 = rep(c("A", "B", "C", "D"), each = 25)) ; set.seed(NULL) ; y = "a" ; categ = c("group1", "group2") ; categ.class.order = list(NULL, c("B", "A", "D", "C")) ; categ.legend.name = "LEGEND" ; categ.color = NULL ; bar.width = 0.8 ; error.disp = "SD" ; error.whisker.width = 0.5 ; dot.color = "same" ; dot.tidy = TRUE ; dot.bin.nb = 60 ; dot.jitter = 0.25 ; dot.size = 3.5 ; dot.border.size = 0.2 ; dot.alpha = 1 ; ylim= c(-15, 25) ; ylog = FALSE ; y.break.nb = NULL ; y.include.zero = FALSE ; y.top.extra.margin = 0.05 ; y.bottom.extra.margin = 0 ; stat.disp = "above" ; stat.size = 4 ; stat.dist = 2 ; xlab = "GROUP" ; ylab = "MEAN" ; vertical = FALSE ; title = "GRAPH1" ; text.size = 20 ; return = TRUE ; classic = FALSE ; grid = FALSE ; return = FALSE ; path.lib = NULL*

*# set.seed(1) ; data1 <- data.frame(a = 1:1000, group1 = rep(c("G", "H"), times = 500), group2 = rep(LETTERS[1:5], each = 200)) ; set.seed(NULL) ; y = "a" ; categ <- c("group1", "group2") ; categ.class.order = list(NULL, c("B", "A", "D", "C", "E")) ; categ.legend.name = "LEGEND" ; categ.color = NULL ; bar.width = 0.8 ; error.disp = "SD" ; error.whisker.width = 1 ; dot.color = NULL ; dot.tidy = FALSE ; dot.bin.nb = 60 ; dot.jitter = 0.25 ; dot.size = 3.5 ; dot.border.size = 0.2 ; dot.alpha = 1 ; ylim= c(-15, 25) ; ylog = FALSE ; y.break.nb = NULL ; y.include.zero = FALSE ; y.top.extra.margin = 0.05 ; y.bottom.extra.margin = 0 ; stat.disp = "above" ; stat.size = 4 ; stat.dist = 1 ; xlab = "GROUP" ; ylab = "MEAN" ; vertical = TRUE ; title = "GRAPH1" ; text.size = 20 ; return = TRUE ; classic = FALSE ; grid = FALSE ; return = FALSE ; path.lib = NULL*

*# function name*

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

*# end function name*

*# required function checking*

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

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

**stop(tempo.cat)**

**}**

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

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

**stop(tempo.cat)**

**}**

**if(length(find("fun\_gg\_palette", mode = "function")) == 0){**

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

**stop(tempo.cat)**

**}**

**if(length(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)**

**}**

**if(length(find("fun\_2D\_comp", mode = "function")) == 0){**

**tempo.cat <- paste0("\n\n================\n\nERROR IN ", function.name, ": REQUIRED fun\_2D\_comp() FUNCTION IS MISSING IN THE R ENVIRONMENT\n\n================\n\n")**

**stop(tempo.cat)**

**}**

*# end required function checking*

*# argument checking (and modification for proper color management)*

**warning <- NULL**

**arg.check <- NULL** *# for function debbuging*

**checked.arg.names <- NULL** *# for function debbuging*

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

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

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

**if(tempo$problem == FALSE & ! (y %in% names(data1))){**

**tempo.cat <- paste0("\n\n================\n\nERROR IN ", function.name, ": y ARGUMENT MUST BE A COLUMN NAME OF data1\n\n================\n\n")**

**stop(tempo.cat)**

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

**if(any(y %in% c("x", "y", "ymax", "x.y", "group"))){**

**tempo.warning <- paste0("RISK OF BUG BECAUSE NAMES IN y ARGUMENT ARE ALSO USED BY FUNCTIONS LIKE merge() OR aggregate()\nIT IS RECOMMENDED TO AVOID THESE COLUMN NAMES IN data1: \"x\", \"y\", \"ymax\", \"x.y\", \"group\"")**

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

**}**

**tempo <- fun\_param\_check(data = data1[, y], data.name = "y COLUMN OF data1", class = "vector", mode = "numeric", na.contain = TRUE, fun.name = function.name) ; eval(ee)**

**}**

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

**if(tempo$problem == FALSE & length(categ) > 2){**

**tempo.cat <- paste0("\n\n================\n\nERROR IN ", function.name, ": categ ARGUMENT CANNOT HAVE MORE THAN 2 COLUMN NAMES OF data1\n\n================\n\n")**

**stop(tempo.cat)**

**}else if(tempo$problem == FALSE & ! all(categ %in% names(data1))){**

**tempo.cat <- paste0("\n\n================\n\nERROR IN ", function.name, ": categ ARGUMENT MUST BE COLUMN NAMES OF data1. HERE IT IS:\n", paste(categ, collapse = " "), "\n\n================\n\n")**

**stop(tempo.cat)**

**}else if(any(categ %in% c("x", "y", "ymax", "x.y", "group"))){**

**tempo.warning <- paste0("RISK OF BUG BECAUSE NAMES IN categ ARGUMENT ARE ALSO USED BY FUNCTIONS LIKE merge() OR aggregate()\nIT IS RECOMMENDED TO AVOID THESE COLUMN NAMES IN data1: \"x\", \"y\", \"ymax\", \"x.y\", \"group\"")**

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

**}**

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

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

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

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

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

**tempo.warning <- paste0("NA DETECTED IN COLUMN ", paste(c(y, categ), collapse = " "), " OF data1 AND CORRESPONDING ROWS REMOVED (SEE $removed.row.nb AND $removed.rows)")**

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

**}else{**

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

**removed.rows <- NULL**

**}**

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

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

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

**tempo.warning <- paste0("IN categ NUMBER ", i1, " IN data1, THE CATEGORY COLUMN ", categ[i1], " CONTAINS NA")**

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

**}**

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

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

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

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

**stop(tempo.cat)**

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

**tempo.warning <- paste0("IN categ NUMBER ", i1, " IN data1, THE CHARACTER COLUMN HAS BEEN CONVERTED TO FACTOR")**

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

**}**

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

**}**

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

**tempo <- fun\_param\_check(data = categ.class.order, class = "list", fun.name = function.name) ; eval(ee)**

**if(tempo$problem == FALSE & length(categ.class.order) > 2){**

**tempo.cat <- paste0("\n\n================\n\nERROR IN ", function.name, ": categ.class.order ARGUMENT MUST BE A LIST OF MAX LENGTH 2\n\n================\n\n")**

**stop(tempo.cat)**

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

**for(i3 in 1:length(categ.class.order)){**

**if(is.null(categ.class.order[[i3]])){**

**tempo.warning <- paste0("THE categ.class.order COMPARTMENT ", i3, " IS NULL. ALPHABETICAL ORDER WILL BE APPLIED")**

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

**data1[, categ[i3]] <- factor(as.character(data1[, categ[i3]]))** *# if already a factor, change nothing, if characters, levels according to alphabetical order*

**}else if(any(duplicated(categ.class.order[[i3]]))){**

**tempo.cat <- paste0("\n\n================\n\nERROR IN ", function.name, ": COMPARTMENT ", i3, " OF categ.class.order ARGUMENT CANNOT HAVE DUPLICATED CLASSES: ", paste(categ.class.order[[i3]], collapse = " "), "\n\n================\n\n")**

**stop(tempo.cat)**

**}else if( ! (all(categ.class.order[[i3]] %in% unique(data1[, categ[i3]])) & all(unique(data1[, categ[i3]]) %in% categ.class.order[[i3]]))){**

**tempo.cat <- paste0("\n\n================\n\nERROR IN ", function.name, ": COMPARTMENT ", i3, " OF categ.class.order ARGUMENT MUST BE CLASSES OF ELEMENT ", i3, " OF categ\nHERE IT IS:\nCOMPARTMENT ", i3, " OF categ.class.order:", paste(categ.class.order[[i3]], collapse = " "), "\nCOLUMN ", categ[i3], " OF data1: ", paste( unique(data1[, categ[i3]]), collapse = " "), "\n\n================\n\n")**

**stop(tempo.cat)**

**}else{**

**data1[, categ[i3]] <- factor(data1[, categ[i3]], levels = categ.class.order[[i3]])** *# reorder the factor*

**}**

**}**

**}**

**}**

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

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

**}else{**

**categ.legend.name <- categ[length(categ)]** *# if only categ1, then legend name of categ1, if length(categ) == 2, then legend name of categ2*

**}**

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

*# check the nature of color*

**tempo1 <- fun\_param\_check(data = categ.color, class = "vector", mode = "character", na.contain = TRUE, fun.name = function.name, print = FALSE)**

**tempo2 <- fun\_param\_check(data = categ.color, class = "factor", na.contain = TRUE, fun.name = function.name, print = FALSE)**

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

*# integer colors into gg\_palette*

**tempo.check.color <- fun\_param\_check(data = categ.color, class = "integer", double.as.integer.allowed = TRUE, na.contain = TRUE, fun.name = function.name, print = FALSE)$problem**

**if(tempo.check.color == TRUE){**

**tempo.cat <- paste0("\n\n================\n\nERROR IN ", function.name, ": categ.color MUST BE A FACTOR OR CHARACTER VECTOR OR INTEGER VECTOR\n\n================\n\n")** *# integer possible because dealt above*

**stop(tempo.cat)**

**}else{** *# convert integers into colors*

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

**}**

*# end integer colors into gg\_palette*

**}**

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

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

**stop(tempo.cat)**

**}**

**if(any(is.na(categ.color))){**

**tempo.warning <- paste0("categ.color ARGUMENT CONTAINS NA")**

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

**}**

*# end check the nature of color*

*# check the length of color*

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

**i0 <- length(categ)** *# if only categ1, then colors for classes of categ1, if length(categ) == 2, then colors for classes of categ2*

**if(length(categ.color) == length(unique(data1[, categ[i0]]))){** *# here length(categ.color) is equal to the different number of categ*

**data1[, categ[i0]] <- factor(data1[, categ[i0]])** *# if already a factor, change nothing, if characters, levels according to alphabetical order*

**data1 <- data.frame(data1, categ.color = data1[, categ[i0]])**

**levels(data1$categ.color) <- categ.color**

**tempo.warning <- paste0("IN ", categ[i0], " OF categ ARGUMENT, THE FOLLOWING COLORS:\n", paste(categ.color, collapse = " "), "\nHAVE BEEN ATTRIBUTED TO THESE CLASSES:\n", paste(levels(factor(data1[, categ[i0]])), collapse = " "))**

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

**}else if(length(categ.color) == length(data1[, categ[i0]])){***# here length(categ.color) is equal to nrow(data1) -> Modif to have length(categ.color) equal to the different number of categ (length(categ.color) == length(levels(data1[, categ[i0]])))*

**data1 <- data.frame(data1, categ.color = categ.color)**

**tempo.check <- unique(data1[ , c(categ[i0], "categ.color")])**

**if( ! (nrow(tempo.check) == length(unique(categ.color)) & nrow(tempo.check) == length(unique(data1[ , categ[i0]])))){**

**tempo.cat <- paste0("\n\n================\n\nERROR IN ", function.name, ": categ.color ARGUMENT HAS THE LENGTH OF data1 ROW NUMBER\nBUT IS INCORRECTLY ASSOCIATED TO EACH CLASS OF categ ", categ[i0], ":\n", paste(unique(mapply(FUN = "paste", data1[ ,categ[i0]], data1[ ,"categ.color"])), collapse = "\n"), "\n\n================\n\n")**

**stop(tempo.cat)**

**}else{**

**data1[, categ[i0]] <- factor(data1[, categ[i0]])** *# if already a factor, change nothing, if characters, levels according to alphabetical order*

**categ.color <- unique(categ.color[order(data1[, categ[i0]])])** *# Modif to have length(categ.color) equal to the different number of categ (length(categ.color) == length(levels(data1[, categ[i0]])))*

**tempo.warning <- paste0("categ.color ARGUMENT HAS THE LENGTH OF data1 ROW NUMBER\nCOLORS HAVE BEEN RESPECTIVELY ASSOCIATED TO EACH CLASS OF categ ", categ[i0], " AS:\n", paste(levels(factor(data1[, categ[i0]])), collapse = " "), "\n", paste(categ.color, collapse = " "))**

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

**}**

**}else if(length(categ.color) == 1){**

**data1[, categ[i0]] <- factor(data1[, categ[i0]])** *# if already a factor, change nothing, if characters, levels according to alphabetical order*

**data1 <- data.frame(data1, categ.color = categ.color)**

**categ.color <- rep(categ.color, length(levels(data1[, categ[i0]])))**

**tempo.warning <- paste0("categ.color ARGUMENT HAS LENGTH 1, MEANING THAT ALL THE DIFFERENT CLASSES OF ", categ[i0], "\n", paste(levels(factor(data1[, categ[i0]])), collapse = " "), "\nWILL HAVE THE SAME COLOR\n", paste(categ.color, collapse = " "))**

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

**}else{**

**tempo.cat <- paste0("\n\n================\n\nERROR IN ", function.name, ": categ.color ARGUMENT MUST BE (1) LENGTH 1, OR (2) THE LENGTH OF data1 NROWS, OR (3) THE LENGTH OF THE CLASSES IN THE categ ", categ[i0], " COLUMN. HERE IT IS COLOR LENGTH ", length(categ.color), " VERSUS CATEG LENGTH ", length(data1[, categ[i0]]), " AND CATEG CLASS LENGTH ", length(unique(data1[, categ[i0]])), "\nPRESENCE OF NA COULD BE THE PROBLEM\n\n================\n\n")**

**stop(tempo.cat)**

**}**

**}else{**

**i0 <- length(categ)** *# if only categ1, then colors for classes of categ1, if length(categ) == 2, then colors for classes of categ2*

**data1[, categ[i0]] <- factor(data1[, categ[i0]])** *# if already a factor, change nothing, if characters, levels according to alphabetical order*

**categ.color <- fun\_gg\_palette(length(levels(data1[, categ[i0]])))**

**data1 <- data.frame(data1, categ.color = data1[, categ[i0]])**

**levels(data1$categ.color) <- categ.color**

**tempo.warning <- paste0("NULL categ.color ARGUMENT -> COLORS RESPECTIVELY ATTRIBUTED TO EACH CLASS OF ", categ[i0], " IN data1:\n", paste(categ.color, collapse = " "), "\n", paste(levels(data1[, categ[i0]]), collapse = " "))**

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

**}**

**tempo <- fun\_param\_check(data = bar.width, prop = TRUE, length = 1, fun.name = function.name) ; eval(ee)**

**if( ! is.null(error.disp)){**

**tempo <- fun\_param\_check(data = error.disp, options = c("SD", "SD.TOP", "SEM", "SEM.TOP"), length = 1, fun.name = function.name) ; eval(ee)**

**}**

**tempo <- fun\_param\_check(data = error.whisker.width, prop = TRUE, length = 1, fun.name = function.name) ; eval(ee)**

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

*# check the nature of color*

**tempo1 <- fun\_param\_check(data = dot.color, class = "vector", mode = "character", na.contain = TRUE, fun.name = function.name, print = FALSE)**

**tempo2 <- fun\_param\_check(data = dot.color, class = "factor", na.contain = TRUE, fun.name = function.name, print = FALSE)**

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

*# integer colors into gg\_palette*

**tempo.check.color <- fun\_param\_check(data = dot.color, class = "integer", double.as.integer.allowed = TRUE, na.contain = TRUE, fun.name = function.name, print = FALSE)$problem**

**if(tempo.check.color == TRUE){**

**tempo.cat <- paste0("\n\n================\n\nERROR IN ", function.name, ": dot.color MUST BE A FACTOR OR CHARACTER VECTOR OR INTEGER VECTOR\n\n================\n\n")** *# integer possible because dealt above*

**stop(tempo.cat)**

**}else{** *# convert integers into colors*

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

**}**

*# end integer colors into gg\_palette*

**}**

**if(all(dot.color == "same") & length(dot.color) == 1){**

**dot.color <- categ.color** *# same color of the dots as the corresponding bar color*

**tempo.warning <- paste0("dot.color ARGUMENT HAS BEEN SET TO \"SAME\"\nTHUS, DOT COLORS HAVE BEEN RESPECTIVELY ASSOCIATED TO EACH CLASS OF categ ", categ[i0], " AS:\n", paste(levels(factor(data1[, categ[i0]])), collapse = " "), "\n", paste(levels(factor(dot.color)), collapse = " "))**

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

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

**tempo.cat <- paste0("\n\n================\n\nERROR IN ", function.name, ": dot.color ARGUMENT MUST BE (1) A HEXADECIMAL COLOR VECTOR STARTING BY #, OR (2) COLOR NAMES GIVEN BY colors(), OR (3) INTEGERS, OR THE STRING\"same\"\nHERE IT IS: ", paste(unique(dot.color), collapse = " "), "\n\n================\n\n")**

**stop(tempo.cat)**

**}**

**if(any(is.na(dot.color))){**

**tempo.warning <- paste0("dot.color ARGUMENT CONTAINS NA")**

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

**}**

*# end check the nature of color*

*# check the length of color*

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

**i0 <- length(categ)** *# if only categ1, then colors for classes of categ1, if length(categ) == 2, then colors for classes of categ2*

**if(length(dot.color) == length(unique(data1[, categ[i0]]))){** *# here length(dot.color) is equal to the different number of categ*

**data1[, categ[i0]] <- factor(data1[, categ[i0]])** *# if already a factor, change nothing, if characters, levels according to alphabetical order*

**data1 <- data.frame(data1, dot.color = data1[, categ[i0]])**

**levels(data1$dot.color) <- dot.color**

**tempo.warning <- paste0("IN ", categ[i0], " OF categ ARGUMENT, THE FOLLOWING COLORS:\n", paste(dot.color, collapse = " "), "\nHAVE BEEN ATTRIBUTED TO THESE CLASSES:\n", paste(levels(factor(data1[, categ[i0]])), collapse = " "))**

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

**}else if(length(dot.color) == length(data1[, categ[i0]])){***# here length(dot.color) is equal to nrow(data1) -> Modif to have length(dot.color) equal to the different number of categ (length(dot.color) == length(levels(data1[, categ[i0]])))*

**data1 <- data.frame(data1, dot.color = dot.color)**

**}else if(length(dot.color) == 1 & ! all(dot.color == "same")){**

**data1[, categ[i0]] <- factor(data1[, categ[i0]])** *# if already a factor, change nothing, if characters, levels according to alphabetical order*

**data1 <- data.frame(data1, dot.color = dot.color)**

**dot.color <- rep(dot.color, length(levels(data1[, categ[i0]])))**

**tempo.warning <- paste0("dot.color ARGUMENT HAS LENGTH 1, MEANING THAT ALL THE DIFFERENT CLASSES OF ", categ[i0], "\n", paste(levels(factor(data1[, categ[i0]])), collapse = " "), "\nWILL HAVE THE SAME COLOR\n", paste(dot.color, collapse = " "))**

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

**}else{**

**tempo.cat <- paste0("\n\n================\n\nERROR IN ", function.name, ": dot.color ARGUMENT MUST BE (1) LENGTH 1, OR (2) THE LENGTH OF data1 NROWS, OR (3) THE LENGTH OF THE CLASSES IN THE categ ", categ[i0], " COLUMN. HERE IT IS COLOR LENGTH ", length(dot.color), " VERSUS CATEG LENGTH ", length(data1[, categ[i0]]), " AND CATEG CLASS LENGTH ", length(unique(data1[, categ[i0]])), "\nPRESENCE OF NA COULD BE THE PROBLEM\n\n================\n\n")**

**stop(tempo.cat)**

**}**

**}**

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

**tempo <- fun\_param\_check(data = dot.bin.nb, class = "vector", typeof = "integer", length = 1, double.as.integer = TRUE, neg.values = FALSE, fun.name = function.name) ; eval(ee)**

**tempo <- fun\_param\_check(data = dot.jitter, prop = TRUE, length = 1, fun.name = function.name) ; eval(ee)**

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

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

**tempo <- fun\_param\_check(data = dot.alpha, prop = TRUE, length = 1, fun.name = function.name) ; eval(ee)**

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

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

**}**

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

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

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

**}**

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

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

**tempo.warning <- paste0("BOTH ylog AND y.include.zero ARGUMENTS SET TO TRUE -> y.include.zero ARGUMENT RESET TO FALSE")**

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

**}**

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

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

**if( ! is.null(stat.disp)){**

**tempo <- fun\_param\_check(data = stat.disp, options = c("top", "above"), length = 1, fun.name = function.name) ; eval(ee)**

**}**

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

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

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

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

**}**

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

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

**}**

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

**if(tempo$problem == FALSE & ylog == TRUE & vertical == FALSE){**

**ylog <- FALSE**

**tempo.warning <- paste0("BECAUSE OF A BUG IN ggplot2, CANNOT FLIP BARS HORIZONTALLY WITH A YLOG SCALE -> ylog ARGUMENT RESET TO FALSE")**

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

**}**

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

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

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

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

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

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

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

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

**cat(paste0("\n\n============\n\nERROR IN ", function.name, ": \nDIRECTORY PATH INDICATED IN THE path.lib PARAMETER DOES NOT EXISTS: ", path.lib, "\n\n============\n\n"))**

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

**}**

**}**

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

**stop()** *# nothing else because print = TRUE by default in fun\_param\_check()*

**}**

*# 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\_param\_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\_param\_check()*

*# end argument checking (and modification for proper color management)*

*# package checking*

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

*# end package checking*

*# main code*

**if(length(categ) == 1){**

*# new data frames for bar and error bars*

**mean.dataframe <- aggregate(x = data1[y], by = {x.env <- list(data1[, categ[1]]) ; names(x.env) <-categ[1] ; x.env}, FUN = mean, na.rm = TRUE)**

**sd.dataframe <- aggregate(x = data1[y], by = {x.env <- list(data1[, categ[1]]) ; names(x.env) <-categ[1] ; x.env}, FUN = sd, na.rm = TRUE)**

**nb.dataframe <- aggregate(x = data1[y], by = {x.env <- list(data1[, categ[1]]) ; names(x.env) <- categ[1] ; x.env}, FUN = function(x.env2){length(x.env2[ ! is.na(x.env2)])})**

**if( ! all(identical(mean.dataframe[, categ[1]], sd.dataframe[, categ[1]]) & identical(mean.dataframe[, categ[1]], nb.dataframe[, categ[1]]))){**

**tempo.cat <- paste0("\n\n================\n\nERROR IN ", function.name, ": aggregate OUTPUT IS DIFFERENT IN TERM OF CLASS ORDER FOR mean.dataframe, sd.dataframe AND nb.dataframe. CODE HAS TO BE MODIFIED\n\n================\n\n")**

**stop(tempo.cat)**

**}else{**

**sem.dataframe <- sd.dataframe**

**sem.dataframe[, y] <- sd.dataframe[, y] / (nb.dataframe[, y])^0.5**

**}**

*# end new data frames for bar and error bars*

*# data1 check categ order for dots coordinates recovery*

**data1 <- data.frame(data1, categ.check = data1[, categ[1]])**

**data1$categ.check <- as.integer(data1$categ.check)** *# to check that data1[, categ[1]] and dot.coord$group are similar, during merging*

*# end data1 check categ order for dots coordinates recovery*

*# per bar dots coordinates recovery*

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

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

**assign(paste0(tempo.gg.name, tempo.gg.count <- tempo.gg.count + 1), ggplot2::ggplot(data = data1, mapping = ggplot2::aes\_string(x = categ[1], y = y, fill = categ[1])))** *# fill because this is what is used with geom\_bar*

**assign(paste0(tempo.gg.name, tempo.gg.count <- tempo.gg.count + 1), ggplot2::geom\_point(stroke = dot.border.size, size = dot.size, alpha = dot.alpha, pch = 21))**

**assign(paste0(tempo.gg.name, tempo.gg.count <- tempo.gg.count + 1), ggplot2::geom\_boxplot())** *# to easily have the equivalent of the grouped bars*

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

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

**dot.coord <- data.frame(dot.coord[order(dot.coord$group, dot.coord$y), ], y.check = as.double(data1[order(data1$categ.check, data1[, y]), y]), categ.check = data1[order(data1$categ.check, data1[, y]), "categ.check"], dot.color = data1[order(data1$categ.check, data1[, y]), "dot.color"], tempo.categ1 = data1[order(data1$categ.check, data1[, y]), categ[1]])** *# y.check to be sure that the order is the same between the y of data1 and the y of dot.coord*

**names(dot.coord)[names(dot.coord) == "tempo.categ1"] <- categ[1]**

**if( ! identical(dot.coord$y, dot.coord$y.check)){**

**tempo.cat <- paste0("\n\n================\n\nERROR IN ", function.name, ": (dot.coord$y AND dot.coord$y.check) AS WELL AS (dot.coord$group AND dot.coord$categ.check) MUST BE IDENTICAL. CODE HAS TO BE MODIFIED\n\n================\n\n")**

**stop(tempo.cat)**

**}**

**}**

*# end per bar dots coordinates recovery*

**}else if(length(categ) == 2){**

*# new data frames for bar and error bars*

**mean.dataframe <- aggregate(x = data1[y], by = {x.env <- list(data1[, categ[1]], data1[, categ[2]]) ; names(x.env) <- c(categ[1], categ[2]) ; x.env}, FUN = mean, na.rm = TRUE)**

**sd.dataframe <- aggregate(x = data1[y], by = {x.env <- list(data1[, categ[1]], data1[, categ[2]]) ; names(x.env) <- c(categ[1], categ[2]) ; x.env}, FUN = sd, na.rm = TRUE)**

**nb.dataframe <- aggregate(x = data1[y], by = {x.env <- list(data1[, categ[1]], data1[, categ[2]]) ; names(x.env) <- c(categ[1], categ[2]) ; x.env}, FUN = function(x.env2){length(x.env2[ ! is.na(x.env2)])})**

**tempo.check.mean <- mapply(FUN = "paste", mean.dataframe[, categ[1]], mean.dataframe[, categ[2]], sep = "\_")**

**tempo.check.sd <- mapply(FUN = "paste", sd.dataframe[, categ[1]], sd.dataframe[, categ[2]], sep = "\_")**

**tempo.check.nb <- mapply(FUN = "paste", nb.dataframe[, categ[1]], nb.dataframe[, categ[2]], sep = "\_")**

**if( ! all(identical(tempo.check.mean, tempo.check.sd) & identical(tempo.check.mean, tempo.check.nb))){**

**tempo.cat <- paste0("\n\n================\n\nERROR IN ", function.name, ": aggregate OUTPUT IS DIFFERENT IN TERM OF CLASS ORDER FOR mean.dataframe, sd.dataframe AND nb.dataframe. CODE HAS TO BE MODIFIED\n\n================\n\n")**

**stop(tempo.cat)**

**}else{**

**sem.dataframe <- sd.dataframe**

**sem.dataframe[, y] <- sd.dataframe[, y] / (nb.dataframe[, y])^0.5**

**}**

*# end new data frames for bar and error bars*

*# data1 check categ order for dots coordinates recovery*

**tempo.factor <- paste0(data1[order(data1[, categ[2]], data1[, categ[1]]), categ[2]], "\_", data1[order(data1[, categ[2]], data1[, categ[1]]), categ[1]])**

**data1 <- data.frame(data1[order(data1[, categ[2]], data1[, categ[1]]), ], categ.check = factor(tempo.factor, levels = unique(tempo.factor)))**

**data1$categ.check <- as.integer(data1$categ.check)**

*# end data1 check categ order for dots coordinates recovery*

*# per bar dots coordinates recovery*

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

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

**assign(paste0(tempo.gg.name, tempo.gg.count <- tempo.gg.count + 1), ggplot2::ggplot(data = data1, mapping = ggplot2::aes\_string(x = categ[1], y = y, fill = categ[2])))** *# fill because this is what is used with geom\_bar*

**assign(paste0(tempo.gg.name, tempo.gg.count <- tempo.gg.count + 1), ggplot2::geom\_point(stroke = dot.border.size, size = dot.size, alpha = dot.alpha, pch = 21))**

**assign(paste0(tempo.gg.name, tempo.gg.count <- tempo.gg.count + 1), ggplot2::geom\_boxplot())** *# to easily have the equivalent of the grouped bars*

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

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

**dot.coord <- data.frame(dot.coord[order(dot.coord$group, dot.coord$y), ], y.check = as.double(data1[order(data1$categ.check, data1[, y]), y]), categ.check = data1[order(data1$categ.check, data1[, y]), "categ.check"], dot.color = data1[order(data1$categ.check, data1[, y]), "dot.color"], tempo.categ1 = data1[order(data1$categ.check, data1[, y]), categ[1]], tempo.categ2 = data1[order(data1$categ.check, data1[, y]), categ[2]])** *# y.check to be sure that the order is the same between the y of data1 and the y of dot.coord*

**names(dot.coord)[names(dot.coord) == "tempo.categ1"] <- categ[1]**

**names(dot.coord)[names(dot.coord) == "tempo.categ2"] <- categ[2]**

**if( ! (identical(dot.coord$y, dot.coord$y.check) & identical(dot.coord$group, dot.coord$categ.check))){**

**tempo.cat <- paste0("\n\n================\n\nERROR IN ", function.name, ": (dot.coord$y AND dot.coord$y.check) AS WELL AS (dot.coord$group AND dot.coord$categ.check) MUST BE IDENTICAL. CODE HAS TO BE MODIFIED\n\n================\n\n")**

**stop(tempo.cat)**

**}**

**}**

**}else{**

**tempo.cat <- (paste0("\n\n============\n\nERROR: CODE INCONSISTENCY\n\n============\n\n"))**

**stop(tempo.cat)**

**}**

**data2 <- mean.dataframe**

**if( ! is.null(error.disp)){**

**if(error.disp == "SD"){**

**data2 <- data.frame(data2, SD = sd.dataframe[, y], ERROR.INF = mean.dataframe[, y] - sd.dataframe[, y], ERROR.SUP = mean.dataframe[, y] + sd.dataframe[, y])**

**}else if(error.disp == "SD.TOP"){**

**data2 <- data.frame(data2, SD = sd.dataframe[, y], ERROR.INF = mean.dataframe[, y], ERROR.SUP = mean.dataframe[, y] + sd.dataframe[, y])**

**}else if(error.disp == "SEM"){**

**data2 <- data.frame(data2, SEM = sem.dataframe[, y], ERROR.INF = mean.dataframe[, y] - sem.dataframe[, y], ERROR.SUP = mean.dataframe[, y] + sem.dataframe[, y])**

**}else if(error.disp == "SEM.TOP"){**

**data2 <- data.frame(data2, SEM = sem.dataframe[, y], ERROR.INF = mean.dataframe[, y], ERROR.SUP = mean.dataframe[, y] + sem.dataframe[, y])**

**}else{**

**tempo.cat <- (paste0("\n\n============\n\nERROR: CODE INCONSISTENCY\n\n============\n\n"))**

**stop(tempo.cat)**

**}**

**}**

*# stat output*

**stat <- data2**

**names(stat)[names(stat) == y] <- "MEAN"**

*# end stat output*

*# range depending on means and error bars*

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

**if(is.null(dot.color)){** *# no dots plotted*

**if( ! is.null(error.disp)){**

**ylim <- range(c(data2[, "ERROR.INF"], data2[, "ERROR.SUP"]), na.rm = TRUE)**

**}else{**

**ylim <- range(data2[, y], na.rm = TRUE)**

**}**

**}else{**

**ylim <- range(data1[, y], na.rm = TRUE)**

**}**

**}**

*# end range depending on means and error bars*

**ylim <- sort(ylim)**

**ylim[1] <- ylim[1] - abs(ylim[2] - ylim[1]) \* y.bottom.extra.margin**

**ylim[2] <- ylim[2] + abs(ylim[2] - ylim[1]) \* y.top.extra.margin**

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

**ylim <- range(c(ylim, 0), na.rm = TRUE)**

**}**

**if(ylog == TRUE & any(ylim < 0)){**

**tempo.cat <- paste0("\n\n================\n\nERROR IN ", function.name, ": FINAL ylim RANGE SPAN NULL OR NEGATIVE VALUES:", paste(ylim, collapse = " "), "\nWHICH IS IMCOMPATIBLE WITH ylog PARAMETER SET TO TRUE\n\n================\n\n")**

**stop(tempo.cat)**

**}**

*# width commputations*

**if(length(categ) == 2){**

**bar.width2 <- bar.width / length(unique(data1[, categ[length(categ)]]))** *# real width of each bar in x-axis unit, among the set of grouped bar. Not relevant if no grouped bars length(categ) == 1*

**}else if(length(categ) == 1){**

**bar.width2 <- bar.width**

**}else{**

**tempo.cat <- (paste0("\n\n============\n\nERROR: CODE INCONSISTENCY\n\n============\n\n"))**

**stop(tempo.cat)**

**}**

**error.whisker.width <- bar.width \* error.whisker.width** *# real error bar width*

**dot.jitter <- bar.width2 \* dot.jitter** *# real dot.jitter*

*# end width commputations*

*# barplot*

*# constant part*

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

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

**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)){categ[1]}else{xlab}))**

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

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

**if(classic == TRUE){**

**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(**

**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.y = ggplot2::element\_line(colour = "grey75")**

**))**

**}else{**

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

**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"),**

**))**

**}**

**}else{**

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

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

**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")**

**))**

**}**

*# end constant part*

*# barplot and error bars*

**assign(paste0(tempo.gg.name, tempo.gg.count <- tempo.gg.count + 1), ggplot2::geom\_bar(data = data2, mapping = ggplot2::aes\_string(x = categ[1], y = y, fill = categ[length(categ)]), stat = "identity", position = ggplot2::position\_dodge(width = NULL), color = "black", width = bar.width))** *# stat = "identity" because already counted, position = position\_dodge(width = NULL) for grouped bars (width = NULL means no overlap between grouped bars). Please, see explanation in https://stackoverflow.com/questions/34889766/what-is-the-width-argument-in-position-dodge/35102486#35102486*

**assign(paste0(tempo.gg.name, tempo.gg.count <- tempo.gg.count + 1), ggplot2::scale\_discrete\_manual(aesthetics = "fill", name = categ.legend.name, values = as.character(categ.color), guide = ggplot2::guide\_legend(override.aes = list(fill = categ.color))))** *# values are the values of color (which is the border color in geom\_bar. Beware: values = categ.color takes the numbers to make the colors if categ.color is a factor*

**if( ! is.null(error.disp)){**

**assign(paste0(tempo.gg.name, tempo.gg.count <- tempo.gg.count + 1), ggplot2::geom\_errorbar(data = data2, mapping = ggplot2::aes\_string(x = categ[1], group = categ[length(categ)], ymin = "ERROR.INF", ymax = "ERROR.SUP"), position = ggplot2::position\_dodge(width = bar.width), color = "black", width = error.whisker.width))** *# cannot use fill = categ[length(categ)] because not an aesthetic of geom\_errorbar, but if only x = categ[1], wrong x coordinates with grouped bars*

**}**

*# end barplot and error bars*

*# coordinates management (for random plotting and for stat display)*

*# bars*

**bar.coord <- ggplot2::ggplot\_build(eval(parse(text = paste(paste0(tempo.gg.name, 1:tempo.gg.count), collapse = " + "))))$data[[1]]** *# to have the summary statistics of the plot. Here because can be required for stat.disp when just bar are plotted*

*# end bars*

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

*# random dots*

**if(dot.tidy == FALSE){**

**dot.coord.rd1 <- merge(dot.coord, bar.coord[c("fill", "group", "x")], by = intersect("group", "group"), sort = FALSE)** *# rd for random. Send the coord of the bars into the coord data.frame of the dots (in the column x.y). Beware: by = intersect("group", "group") because group is enough as only one value of x per group number in bar.coord. Thus, no need to consider fill*

**if(nrow(dot.coord.rd1) != nrow(dot.coord)){**

**tempo.cat <- paste0("\n\n================\n\nERROR IN ", function.name, ": THE merge() FUNCTION DID NOT RETURN A CORRECT dot.coord.rd1 DATA FRAME. CODE HAS TO BE MODIFIED\n\n================\n\n")**

**stop(tempo.cat)**

**}**

**set.seed(1)**

**sampled.dot.jitter <- if(nrow(dot.coord.rd1) == 1){runif(n = nrow(dot.coord.rd1), min = - dot.jitter / 2, max = dot.jitter / 2)}else{sample(x = runif(n = nrow(dot.coord.rd1), min = - dot.jitter / 2, max = dot.jitter / 2), size = nrow(dot.coord.rd1), replace = FALSE)}**

**dot.coord.rd2 <- data.frame(dot.coord.rd1, dot.x = dot.coord.rd1$x.y + sampled.dot.jitter)** *# set the dot.jitter thanks to runif and dot.jitter range. Then, send the coord of the bars into the coord data.frame of the dots (in the column x.y)*

**set.seed(NULL)**

**if(length(categ) == 1){**

**tempo.data1 <- unique(data.frame(data1[categ[1]], group = as.integer(factor(as.numeric(data1[, categ[1]])))))** *# categ[2] first if categ[2] is used to make the categories in ggplot and categ[1] is used to make the x-axis*

**names(tempo.data1)[names(tempo.data1) == categ[1]] <- paste0(categ[1], ".check")**

**verif <- paste0(categ[1], ".check")**

**}else if(length(categ) == 2){**

**tempo.data1 <- unique(data.frame(data1[c(categ[1], categ[2])], group = as.integer(factor(paste0(as.numeric(data1[, categ[2]]), ".", as.numeric(data1[, categ[1]]))))))** *# categ[2] first if categ[2] is used to make the categories in ggplot and categ[1] is used to make the x-axis*

**names(tempo.data1)[names(tempo.data1) == categ[1]] <- paste0(categ[1], ".check")**

**names(tempo.data1)[names(tempo.data1) == categ[2]] <- paste0(categ[2], ".check")**

**verif <- c(paste0(categ[1], ".check"), paste0(categ[2], ".check"))**

**}else{**

**tempo.cat <- (paste0("\n\n============\n\nERROR: CODE INCONSISTENCY\n\n============\n\n"))**

**stop(tempo.cat)**

**}**

**dot.coord.rd3 <- merge(dot.coord.rd2, tempo.data1, by = "group", sort = FALSE)** *# send the factors of data1 into coord*

**if(nrow(dot.coord.rd3) != nrow(dot.coord) | ( ! fun\_2D\_comp(dot.coord.rd3[categ], dot.coord.rd3[verif])$identical.content)){**

**tempo.cat <- paste0("\n\n================\n\nERROR IN ", function.name, ": THE merge() FUNCTION DID NOT RETURN A CORRECT dot.coord.rd3 DATA FRAME. CODE HAS TO BE MODIFIED\n\n================\n\n")**

**stop(tempo.cat)**

**}**

*# end random dots*

**}**

*# tidy dots*

*# coordinates are recover during plotting (see dot.coord.tidy1 below)*

*# end tidy dots*

**}**

*# end coordinates management (for random plotting and for stat display)*

*# dot display*

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

**if(dot.tidy == FALSE){**

**if(dot.border.size == 0){**

**assign(paste0(tempo.gg.name, tempo.gg.count <- tempo.gg.count + 1), ggplot2::geom\_point(data = dot.coord.rd3, mapping = ggplot2::aes\_string(x = "dot.x", y = "y", group = categ[length(categ)]), size = dot.size, color = dot.coord.rd3$dot.color, alpha = dot.alpha, pch = 16))** *# group used in aesthetic to do not have it in the legend. Here ggplot2::scale\_discrete\_manual() cannot be used because of the group easthetic*

**}else{**

**assign(paste0(tempo.gg.name, tempo.gg.count <- tempo.gg.count + 1), ggplot2::geom\_point(data = dot.coord.rd3, mapping = ggplot2::aes\_string(x = "dot.x", y = "y", group = categ[length(categ)]), stroke = dot.border.size, size = dot.size, fill = dot.coord.rd3$dot.color, alpha = dot.alpha, pch = 21))** *# group used in aesthetic to do not have it in the legend. Here ggplot2::scale\_discrete\_manual() cannot be used because of the group easthetic*

**}**

**}else if(dot.tidy == TRUE){**

**assign(paste0(tempo.gg.name, tempo.gg.count <- tempo.gg.count + 1), ggplot2::geom\_dotplot(data = dot.coord, mapping = ggplot2::aes\_string(x = categ[1], y = "y", color = categ[length(categ)]), position = ggplot2::position\_dodge(width = bar.width), binaxis = "y", stackdir = "center", stroke = dot.border.size, alpha = dot.alpha, fill = dot.coord[rev(order(dot.coord[, categ[1]], decreasing = TRUE)), "dot.color"], show.legend = FALSE, binwidth = (ylim[2] - ylim[1]) / dot.bin.nb))** *# very weird behavior of geom\_dotplot, because data1 seems reorderer according to x = categ[1] before plotting. Thus, I have to use fill = dot.coord[rev(order(dot.coord[, categ[1]], decreasing = TRUE)), "dot.color"] to have the good corresponding colors*

**assign(paste0(tempo.gg.name, tempo.gg.count <- tempo.gg.count + 1), ggplot2::scale\_discrete\_manual(aesthetics = "color", name = categ.legend.name, values = rep("black", length(categ.color))))** *# values = rep("black", length(categ.color)) are the values of color (which is the border color of dots), and this modify the border color on the plot. Beware: values = categ.color takes the numbers to make the colors if categ.color is a factor. BEWARE: , guide = ggplot2::guide\_legend(override.aes = list(fill = levels(dot.color))) here*

*# coordinates of tidy dots*

**tempo.coord <- ggplot2::ggplot\_build(eval(parse(text = paste(paste0(tempo.gg.name, 1:tempo.gg.count), collapse = " + "))))$data** *# to have the tidy dot coordinates*

**if(length(which(sapply(tempo.coord, FUN = nrow) == nrow(data1))) > 1){**

**tempo.cat <- paste0("\n\n================\n\nERROR IN ", function.name, ": MORE THAN 2 COMPARTMENT WITH NROW EQUAL TO nrow(data1) IN THE tempo.coord LIST (FOR TIDY DOT COORDINATES). CODE HAS TO BE MODIFIED\n\n================\n\n")**

**stop(tempo.cat)**

**}else{**

**dot.coord.tidy1 <- tempo.coord[[which(sapply(tempo.coord, FUN = nrow) == nrow(data1))]]**

**}**

**tempo.bar.coord <- merge(bar.coord, unique(dot.coord[, c("group", categ)]), by = intersect("group", "group"), sort = FALSE)** *# add the categ in bar.coord. Beware: by = intersect("group", "group") because group is enough as only one value of x per group number in bar.coord. Thus, no need to consider fill*

**if(nrow(tempo.bar.coord) != nrow(bar.coord)){**

**tempo.cat <- paste0("\n\n================\n\nERROR IN ", function.name, ": THE merge() FUNCTION DID NOT RETURN A CORRECT tempo.bar.coord DATA FRAME. CODE HAS TO BE MODIFIED\n\n================\n\n")**

**stop(tempo.cat)**

**}**

**dot.coord.tidy2 <- merge(dot.coord.tidy1, tempo.bar.coord[c("fill", "group", "x", categ)], by = intersect("group", "group"), sort = FALSE)** *# send the coord of the bars into the coord data.frame of the dots (in the column x.y). Beware: by = intersect("group", "group") because group is enough as only one value of x per group number in bar.coord. Thus, no need to consider fill*

**if(nrow(dot.coord.tidy2) != nrow(dot.coord)){**

**tempo.cat <- paste0("\n\n================\n\nERROR IN ", function.name, ": THE merge() FUNCTION DID NOT RETURN A CORRECT dot.coord.tidy2 DATA FRAME. CODE HAS TO BE MODIFIED\n\n================\n\n")**

**stop(tempo.cat)**

**}**

**if(length(categ) == 1){**

**tempo.data1 <- unique(data.frame(data1[categ[1]], group = as.integer(factor(as.numeric(data1[, categ[1]])))))** *# categ[2] first if categ[2] is used to make the categories in ggplot and categ[1] is used to make the x-axis*

**names(tempo.data1)[names(tempo.data1) == categ[1]] <- paste0(categ[1], ".check")**

**verif <- paste0(categ[1], ".check")**

**}else if(length(categ) == 2){**

**tempo.data1 <- unique(data.frame(data1[c(categ[1], categ[2])], group = as.integer(factor(paste0(as.numeric(data1[, categ[2]]), ".", as.numeric(data1[, categ[1]]))))))** *# categ[2] first if categ[2] is used to make the categories in ggplot and categ[1] is used to make the x-axis*

**names(tempo.data1)[names(tempo.data1) == categ[1]] <- paste0(categ[1], ".check")**

**names(tempo.data1)[names(tempo.data1) == categ[2]] <- paste0(categ[2], ".check")**

**verif <- c(paste0(categ[1], ".check"), paste0(categ[2], ".check"))**

**}else{**

**tempo.cat <- (paste0("\n\n============\n\nERROR: CODE INCONSISTENCY\n\n============\n\n"))**

**stop(tempo.cat)**

**}**

**dot.coord.tidy3 <- merge(dot.coord.tidy2, tempo.data1, by = "group", sort = FALSE)** *# send the factors of data1 into coord*

**if(nrow(dot.coord.tidy3) != nrow(dot.coord) | ( ! fun\_2D\_comp(dot.coord.tidy3[categ], dot.coord.tidy3[verif])$identical.content)){**

**tempo.cat <- paste0("\n\n================\n\nERROR IN ", function.name, ": THE merge() FUNCTION DID NOT RETURN A CORRECT dot.coord.tidy3 DATA FRAME. CODE HAS TO BE MODIFIED\n\n================\n\n")**

**stop(tempo.cat)**

**}**

*# end coordinates of tidy dots*

**}**

**}**

*# end dot display*

*# stat display*

*# layer after dots but ok, behind dots on the plot*

**if( ! is.null(stat.disp)){**

**if(stat.disp == "top"){**

**assign(paste0(tempo.gg.name, tempo.gg.count <- tempo.gg.count + 1), ggplot2::annotate(geom = "text", x = bar.coord$x, y = ylim[2], label = fun\_round(bar.coord$y, 2), size = stat.size, color = "black", hjust = ifelse(vertical == TRUE, 0.5, 1.1), vjust = ifelse(vertical == TRUE, 1.1, 0.5)))** *# beware: no need of order() for labels because bar.coord$x set the order. For justification, see https://stackoverflow.com/questions/7263849/what-do-hjust-and-vjust-do-when-making-a-plot-using-ggplot*

**}else if(stat.disp == "above"){**

*# stat coordinates*

**if( ! is.null(dot.color)){** *# for text just above max dot*

**if(dot.tidy == FALSE){**

**tempo.stat.ini <- dot.coord.rd3**

**}else if(dot.tidy == TRUE){**

**tempo.stat.ini <- dot.coord.tidy3**

**}**

**stat.coord1 <- aggregate(x = tempo.stat.ini["y"], by = {x.env <- if(length(categ) == 1){list(tempo.stat.ini$group, tempo.stat.ini$x.y, tempo.stat.ini[, categ[1]])}else if(length(categ) == 2){list(tempo.stat.ini$group, tempo.stat.ini$x.y, tempo.stat.ini[, categ[1]], tempo.stat.ini[, categ[2]])} ; names(x.env) <- if(length(categ) == 1){c("group", "x.y", categ[1])}else if(length(categ) == 2){c("group", "x.y", categ[1], categ[2])} ; x.env}, FUN = min, na.rm = TRUE)**

**names(stat.coord1)[names(stat.coord1) == "y"] <- "dot.min"**

**stat.coord2 <- aggregate(x = tempo.stat.ini["y"], by = {x.env <- if(length(categ) == 1){list(tempo.stat.ini$group, tempo.stat.ini$x.y, tempo.stat.ini[, categ[1]])}else if(length(categ) == 2){list(tempo.stat.ini$group, tempo.stat.ini$x.y, tempo.stat.ini[, categ[1]], tempo.stat.ini[, categ[2]])} ; names(x.env) <- if(length(categ) == 1){c("group", "x.y", categ[1])}else if(length(categ) == 2){c("group", "x.y", categ[1], categ[2])} ; x.env}, FUN = max, na.rm = TRUE)**

**names(stat.coord2) <- paste0(names(stat.coord2), "\_from.dot.max")**

**names(stat.coord2)[names(stat.coord2) == "y\_from.dot.max"] <- "dot.max"**

**stat.coord3 <- cbind(bar.coord[order(bar.coord$x), ], stat.coord1[order(stat.coord1$x.y), ], stat.coord2[order(stat.coord2$x.y), ])** *# should be ok to use bar.coord$x and stat.coord$x.y to assemble the two data frames because x coordinates of the bars. Thus, we cannot have identical values*

**if( ! all(identical(round(stat.coord3$x, 9), round(stat.coord3$x.y, 9)))){**

**tempo.cat <- paste0("\n\n================\n\nERROR IN ", function.name, ": FUSION OF bar.coord, stat.coord1 AND stat.coord2 ACCORDING TO bar.coord$x, stat.coord1$x.y AND stat.coord2$x.y IS NOT CORRECT. CODE HAS TO BE MODIFIED\n\n================\n\n")**

**stop(tempo.cat)**

**}**

**dot.text.coord <- stat.coord3[, c("x", "group", "dot.min", "dot.max")]**

**names(dot.text.coord)[names(dot.text.coord) == "dot.min"] <- "text.min.pos"**

**names(dot.text.coord)[names(dot.text.coord) == "dot.max"] <- "text.max.pos"**

**}**

**if( ! is.null(error.disp)){** *# for text just above error bars*

**if(length(categ) == 1){**

**tempo.data1 <- unique(data.frame(data1[categ[1]], group = as.integer(factor(as.numeric(data1[, categ[1]])))))** *# categ[2] first if categ[2] is used to make the categories in ggplot and categ[1] is used to make the x-axis*

**if( ! identical(stat[order(stat[, categ[1]]), categ[1]], tempo.data1[order(tempo.data1[, categ[1]]), categ[1]])){**

**tempo.cat <- (paste0("\n\n============\n\nERROR: CODE PROBLEM IN TRYING TO ASSEMBLE stat AND tempo.data1\n\n============\n\n"))**

**stop(tempo.cat)**

**}else{**

**names(tempo.data1)[names(tempo.data1) == categ[1]] <- paste0(categ[1], ".check")**

**names(tempo.data1)[names(tempo.data1) == "group"] <- "group.check"**

**stat.coord4 <- cbind(stat[order(stat[, categ[1]]), ], tempo.data1[order(tempo.data1[, paste0(categ[1], ".check")]), ])**

**}**

**}else if(length(categ) == 2){**

**tempo.data1 <- unique(data.frame(data1[c(categ[1], categ[2])], group = as.integer(factor(paste0(as.numeric(data1[, categ[2]]), ".", as.numeric(data1[, categ[1]]))))))** *# categ[2] first if categ[2] is used to make the categories in ggplot and categ[1] is used to make the x-axis*

**if( ! fun\_2D\_comp(stat[order(stat[, categ[1]], stat[, categ[2]]), c(categ[1], categ[2])], tempo.data1[order(tempo.data1[, categ[1]], tempo.data1[, categ[2]]), c(categ[1], categ[2])])$identical.content){**

**tempo.cat <- (paste0("\n\n============\n\nERROR: CODE PROBLEM IN TRYING TO ASSEMBLE stat AND tempo.data1\n\n============\n\n"))**

**stop(tempo.cat)**

**}else{**

**names(tempo.data1)[names(tempo.data1) == categ[1]] <- paste0(categ[1], ".check")**

**names(tempo.data1)[names(tempo.data1) == categ[2]] <- paste0(categ[2], ".check")**

**names(tempo.data1)[names(tempo.data1) == "group"] <- "group.check"**

**stat.coord4 <- cbind(stat[order(stat[, categ[1]], stat[, categ[2]]), ], tempo.data1[order(tempo.data1[, paste0(categ[1], ".check")], tempo.data1[,paste0(categ[2], ".check")]), ])**

**}**

**}else{**

**tempo.cat <- (paste0("\n\n============\n\nERROR: CODE INCONSISTENCY\n\n============\n\n"))**

**stop(tempo.cat)**

**}**

**if( ! identical(bar.coord$group[order(bar.coord$group)], stat.coord4$group.check[order(stat.coord4$group.check)])){**

**tempo.cat <- (paste0("\n\n============\n\nERROR: CODE PROBLEM IN TRYING TO ASSEMBLE bar.coord AND stat.coord4\n\n============\n\n"))**

**stop(tempo.cat)**

**}else{**

**stat.coord5 <- cbind(bar.coord[order(bar.coord$group), ], stat.coord4[order(stat.coord4$group.check), ])**

**error.text.coord <- stat.coord5[, c("x", "group", "ERROR.INF", "ERROR.SUP")] #**

**names(error.text.coord)[names(error.text.coord) == "ERROR.INF"] <- "text.min.pos"**

**names(error.text.coord)[names(error.text.coord) == "ERROR.SUP"] <- "text.max.pos"**

**}**

**}**

**if(( ! is.null(dot.color)) & ! is.null(error.disp)){** *# for text above max dot or error bar*

**stat.coord3 <- stat.coord3[order(stat.coord3$x), ]**

**stat.coord5 <- stat.coord5[order(stat.coord5$x), ]**

**if( ! identical(stat.coord3$group, stat.coord5$group)){**

**tempo.cat <- (paste0("\n\n============\n\nERROR: CODE PROBLEM IN TRYING TO ASSEMBLE stat.coord3 AND stat.coord5\n\n============\n\n"))**

**stop(tempo.cat)**

**}else{**

**stat.coord6 <- data.frame(stat.coord3, min.dot.error = mapply(FUN = min, stat.coord3$dot.min, stat.coord5$ERROR.INF, na.rm = TRUE))**

**stat.coord7 <- data.frame(stat.coord6, max.dot.error = mapply(FUN = max, stat.coord3$dot.max, stat.coord5$ERROR.SUP, na.rm = TRUE))**

**both.text.coord <- stat.coord7[, c("x", "group", "min.dot.error", "max.dot.error")] #**

**names(both.text.coord)[names(both.text.coord) == "min.dot.error"] <- "text.min.pos"**

**names(both.text.coord)[names(both.text.coord) == "max.dot.error"] <- "text.max.pos"**

**}**

**}**

**if(( ! is.null(dot.color)) & is.null(error.disp)){**

**text.coord <- dot.text.coord**

**}else if(is.null(dot.color) & ! is.null(error.disp)){**

**text.coord <- error.text.coord**

**}else if(( ! is.null(dot.color)) & ! is.null(error.disp)){**

**text.coord <- both.text.coord**

**}**

**if( ! (is.null(dot.color) & is.null(error.disp))){**

**bar.coord <- bar.coord[order(bar.coord$x), ]**

**text.coord <- text.coord[order(text.coord$x), ]** *# to be sure to have the two objects in the same order for x. BEWARE: cannot add identical(as.integer(text.coord$group), as.integer(bar.coord$group)) because with error, the correspondence between x and group is not the same*

**if( ! identical(text.coord$x, bar.coord$x)){**

**tempo.cat <- (paste0("\n\n============\n\nERROR: text.coord AND bar.coord DO NOT HAVE THE SAME x COLUMN CONTENT\n\n============\n\n"))**

**stop(tempo.cat)**

**}**

**}**

*# end stat coordinates*

*# stat display*

**if(is.null(dot.color) & is.null(error.disp)){** *# text just above bars*

*# performed twice: first for y values >=0, then y values < 0, because only a single value allowed for hjust anf vjust*

**assign(paste0(tempo.gg.name, tempo.gg.count <- tempo.gg.count + 1), ggplot2::annotate(geom = "text", x = bar.coord$x[bar.coord$y >= 0], y = bar.coord$y[bar.coord$y >= 0], label = fun\_round(bar.coord$y, 2)[bar.coord$y >= 0], size = stat.size, color = "black", hjust = ifelse(vertical == TRUE, 0.5, 0.5 - stat.dist), vjust = ifelse(vertical == TRUE, 0.5 - stat.dist, 0.5)))** *# beware: no need of order() for labels because bar.coord$x set the order*

**assign(paste0(tempo.gg.name, tempo.gg.count <- tempo.gg.count + 1), ggplot2::annotate(geom = "text", x = bar.coord$x[bar.coord$y < 0], y = bar.coord$y[bar.coord$y < 0], label = fun\_round(bar.coord$y, 2)[bar.coord$y < 0], size = stat.size, color = "black", hjust = ifelse(vertical == TRUE, 0.5, 0.5 + stat.dist), vjust = ifelse(vertical == TRUE, 0.5 + stat.dist, 0.5)))** *# beware: no need of order() for labels because bar.coord$x set the order*

**}else{** *# text just above error bars or dots*

*# I checked that text.coord and bar.coord have the same x and group column content. Thus, ok to use them together*

**assign(paste0(tempo.gg.name, tempo.gg.count <- tempo.gg.count + 1), ggplot2::annotate(geom = "text", x = text.coord$x[bar.coord$y >= 0], y = text.coord$text.max.pos[bar.coord$y >= 0], label = fun\_round(bar.coord$y, 2)[bar.coord$y >= 0], size = stat.size, color = "black", hjust = ifelse(vertical == TRUE, 0.5, 0.5 - stat.dist), vjust = ifelse(vertical == TRUE, 0.5 - stat.dist, 0.5)))** *# beware: no need of order() for labels because bar.coord$x set the order*

**assign(paste0(tempo.gg.name, tempo.gg.count <- tempo.gg.count + 1), ggplot2::annotate(geom = "text", x = text.coord$x[bar.coord$y < 0], y = text.coord$text.min.pos[bar.coord$y < 0], label = fun\_round(bar.coord$y, 2)[bar.coord$y < 0], size = stat.size, color = "black", hjust = ifelse(vertical == TRUE, 0.5, 0.5 + stat.dist), vjust = ifelse(vertical == TRUE, 0.5 + stat.dist, 0.5)))** *# beware: no need of order() for labels because bar.coord$x set the order*

**}**

*# end stat display*

**}else{**

**tempo.cat <- (paste0("\n\n============\n\nERROR: CODE INCONSISTENCY\n\n============\n\n"))**

**stop(tempo.cat)**

**}**

**}**

*# end stat display*

*# y scale management (cannot be before dot plot management)*

**if(ylog == TRUE){**

**assign(paste0(tempo.gg.name, tempo.gg.count <- tempo.gg.count + 1), ggplot2::annotation\_logticks(sides = "l"))** *# string containing any of "trbl", for top, right, bottom, and left*

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

**assign(paste0(tempo.gg.name, tempo.gg.count <- tempo.gg.count + 1), ggplot2::scale\_y\_continuous(breaks = fun\_round(seq(ylim[1], ylim[2], length.out = y.break.nb), dec.nb = 2, after.lead.zero = TRUE)))**

**}**

**}else{**

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

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

**breaks = fun\_round(seq(ylim[1], ylim[2], length.out = y.break.nb), dec.nb = 2, after.lead.zero = TRUE),**

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

**limits = NA**

**))**

**}else{**

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

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

**limits = NA**

**))**

**}**

**}**

**if(vertical == TRUE){**

**assign(paste0(tempo.gg.name, tempo.gg.count <- tempo.gg.count + 1), ggplot2::coord\_cartesian(ylim = ylim))**

**}else{**

**assign(paste0(tempo.gg.name, tempo.gg.count <- tempo.gg.count + 1), ggplot2::coord\_flip(ylim = ylim))**

**}**

*# end y scale management (cannot be before dot plot management)*

**suppressWarnings(print(eval(parse(text = paste(paste0(tempo.gg.name, 1:tempo.gg.count), collapse = " + ")))))**

*# end barplot*

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

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

**output <- list(stat = stat, removed.row.nb = removed.row.nb, removed.rows = removed.rows, data = output$data, warnings = paste0("\n", warning, "\n\n"))**

**return(output)**

**}**

**}**

######## fun\_gg\_boxplot() #### ggplot2 boxplot + background dots if required

**# http://www.sthda.com/english/wiki/ggplot2-box-plot-quick-start-guide-r-software-and-data-visualization**

**fun\_gg\_boxplot <- function(data1, y, categ, class.order = NULL, legend.name = NULL, categ.color = NULL, dot.color = "same", box.width = 0.5, whisker.width = 0.5, jitter = 0.25, ylim = NULL, ylog = FALSE, y.include.zero = FALSE, top.extra.margin = 0.05, bottom.extra.margin = 0, xlab = NULL, ylab = NULL, pt.size = 3, pt.border.size = 0.5, alpha = 0.5, show.stat = NULL, stat.size = 4, title = "", text.size = 12, break.nb = NULL, classic = FALSE, grid = FALSE, return = FALSE, path.lib = 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)*

*# to have a single boxplot, 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 boxplots, 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 boxplots (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\_mean() 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\_mean() output report the bar boundaries (around x-axis unit 1, 2, 3, etc., for each set of grouped bar)*

*# ARGUMENTS*

*# data1: a dataframe containing one column of values (see y argument below) and one or two columns of categories (see categ argument below)*

*# y: character string of the data1 column name for y-axis (containing numeric values). Numeric values will be used to generate the boxplots 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 boxplot per class of categ1. If two column names (further refered to as categ1 and categ2), then one boxplot per class of categ2, which form a group of boxplots 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 boxplot, 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 boxplots, create a factor column with a single class and specify this column in categ argument as first element (categ1)*

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

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

*# categ.color: vector of character color string for boxplot color. 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 boxplots 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 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 boxplots inside a group of boxplots)*

*# 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*

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

*# 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*

*# 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*

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

*# ylog: logical. Log10 scale for the y-axis? Beware: if TRUE, ylim must not contain null or negative values*

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

*# 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 \* top.extra.margin (e.g., abs(ylim[2] - ylim[1]) \* top.extra.margin) to the top of y-axis. Beware with ylog = TRUE, the range result must not overlap zero or negative values*

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

*# 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*

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

*# pt.border.size: numeric value of border dot size. Write zero for no stroke*

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

*# show.stat: 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 number size (in points)*

*# title: character string of the graph title*

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

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

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

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

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

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

*# REQUIRED PACKAGES*

*# ggplot2*

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

*# fun\_param\_check()*

*# fun\_pack\_import()*

*# fun\_gg\_palette()*

*# fun\_round()*

*# fun\_2D\_comp()*

*# RETURN*

*# a boxplot*

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

*# stat: the graphic statistics*

*# removed.row.nb: which rows have been removed due to NA detection in y and categ columns (NULL if no row removed)*

*# removed.rows: removed rows containing NA (NULL if no row removed)*

*# data: the graphic info coordinates*

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

*# EXAMPLES*

*# obs1 <- data.frame(a = 1:20, group1 = rep(c("G", "H"), times = 10)) ; fun\_gg\_bar\_mean(data1 = obs1, y = "a", categ = "group1", categ.color = "white") # separate bars, modification of bar color 1 (a single value)*

*# obs1 <- data.frame(a = 1:20, group1 = rep(c("G", "H"), times = 10)) ; fun\_gg\_bar\_mean(data1 = obs1, y = "a", categ = "group1", categ.color = c("red", "blue")) # separate bars, modification of bar color 2 (one value par class of categ2)*

*# obs1 <- data.frame(a = 1:20, group1 = rep(c("G", "H"), times = 10), bar.color = rep(c("brown", "orange"), time = 10)) ; obs1 ; fun\_gg\_bar\_mean(data1 = obs1, y = "a", categ = "group1", categ.color = obs1$bar.color) # separate bars, modification of bar color 3 (one value per line of obs1, with respect of the correspondence between categ2 and bar.color columns)*

*# obs1 <- data.frame(a = 1:20, group1 = rep(c("G", "H"), times = 10)) ; fun\_gg\_bar\_mean(data1 = obs1, y = "a", categ = "group1", dot.color = "same") # separate bars, modification of dot color 1 (same dot color as the corresponding bar)*

*# obs1 <- data.frame(a = 1:20, group1 = rep(c("G", "H"), times = 10)) ; fun\_gg\_bar\_mean(data1 = obs1, y = "a", categ = "group1", dot.color = "green") # separate bars, modification of dot color 2 (single color for all the dots)*

*# obs1 <- data.frame(a = 1:20, group1 = rep(c("G", "H"), times = 10)) ; fun\_gg\_bar\_mean(data1 = obs1, y = "a", categ = "group1", dot.color = c("green", "brown")) # separate bars, modification of dot color 3 (one value par class of categ2)*

*# obs1 <- data.frame(a = 1:10, group1 = rep(c("G", "H"), times = 5)) ; fun\_gg\_bar\_mean(data1 = obs1, y = "a", categ = "group1", dot.color = hsv(h = (1:nrow(obs1)) / nrow(obs1))) # separate bars, modification of dot color 4 (any color for each dot)*

*# obs1 <- data.frame(a = 1:20, group1 = rep(c("G", "H"), times = 10), group2 = rep(c("A", "B"), each = 10)) ; fun\_gg\_bar\_mean(data1 = obs1, y = "a", categ = c("group1", "group2")) # grouped bars, default arguments*

*# obs1 <- data.frame(a = 1:24, group1 = rep(c("G", "H"), times = 12), group2 = rep(c("A", "B", "C", "D"), each = 6)) ; fun\_gg\_bar\_mean(data1 = obs1, y = "a", categ = c("group1", "group2"), return = TRUE) # more grouped bars*

*# obs1 <- data.frame(a = log10((1:20) \* 100), group1 = rep(c("G", "H"), times = 10), group2 = rep(c("A", "B"), each = 10)) ; fun\_gg\_bar\_mean(data1 = obs1, y = "a", categ = c("group1", "group2"), ylog = TRUE) # grouped bars, log scale. Beware, y column must be log, otherwise incoherent scale*

*# obs1 <- data.frame(a = 1:20, group1 = rep(c("G", "H"), times = 10), group2 = rep(c("A", "B"), each = 10)) ; fun\_gg\_bar\_mean(data1 = obs1, y = "a", categ = c("group1", "group2"), dot.color = NULL) # grouped bars, no dots*

*# obs1 <- data.frame(a = 1:20, group1 = rep(c("G", "H"), times = 10), group2 = rep(c("A", "B"), each = 10)) ; fun\_gg\_bar\_mean(data1 = obs1, y = "a", categ = c("group1", "group2"), categ.color = "white") # grouped bars, modification of bar color 1 (a single value)*

*# obs1 <- data.frame(a = 1:20, group1 = rep(c("G", "H"), times = 10), group2 = rep(c("A", "B"), each = 10)) ; fun\_gg\_bar\_mean(data1 = obs1, y = "a", categ = c("group1", "group2"), categ.color = c("red", "blue")) # grouped bars, modification of bar color 2 (one value par class of categ2)*

*# obs1 <- data.frame(a = 1:20, group1 = rep(c("G", "H"), times = 10), group2 = rep(c("A", "B"), each = 10), bar.color = rep(c("brown", "orange"), each = 10)) ; obs1 ; fun\_gg\_bar\_mean(data1 = obs1, y = "a", categ = c("group1", "group2"), categ.color = obs1$bar.color) # grouped bars, modification of bar color 3 (one value per line of obs1, with respect of the correspondence between categ2 and bar.color columns)*

*# obs1 <- data.frame(a = 1:20, group1 = rep(c("G", "H"), times = 10), group2 = rep(c("A", "B"), each = 10)) ; fun\_gg\_bar\_mean(data1 = obs1, y = "a", categ = c("group1", "group2"), dot.color = "same") # grouped bars, modification of dot color 1 (same dot color as the corresponding bar)*

*# obs1 <- data.frame(a = 1:20, group1 = rep(c("G", "H"), times = 10), group2 = rep(c("A", "B"), each = 10)) ; fun\_gg\_bar\_mean(data1 = obs1, y = "a", categ = c("group1", "group2"), dot.color = "green") # grouped bars, modification of dot color 2 (single color for all the dots)*

*# obs1 <- data.frame(a = 1:20, group1 = rep(c("G", "H"), times = 10), group2 = rep(c("A", "B"), each = 10)) ; fun\_gg\_bar\_mean(data1 = obs1, y = "a", categ = c("group1", "group2"), dot.color = c("green", "brown")) # grouped bars, modification of dot color 3 (one value par class of categ2)*

*# obs1 <- data.frame(a = 1:10, group1 = rep(c("G", "H"), times = 5), group2 = rep(c("A", "B"), each = 5)) ; fun\_gg\_bar\_mean(data1 = obs1, y = "a", categ = c("group1", "group2"), dot.color = hsv(h = (1:nrow(obs1)) / nrow(obs1))) # grouped bars, modification of dot color 4 (any color for each dot)*

*# obs1 <- data.frame(a = 1:20, group1 = rep(c("G", "H"), times = 10), group2 = rep(c("A", "B"), each = 10)) ; fun\_gg\_bar\_mean(data1 = obs1, y = "a", categ = c("group1", "group2"), class.order = list(NULL, c("B", "A")), legend.name = "", categ.color = c("red", "blue"), dot.color = "grey", error.bar = "SD", bar.width = 0.25, error.bar.width = 0.8, jitter = 1, ylim = c(10, 30), y.include.zero = FALSE, top.extra.margin = 0.5, bottom.extra.margin = 1, xlab = "GROUP", ylab = "MEAN", pt.size = 4, pt.border.size = 0, alpha = 1, show.stat = "above", stat.size = 4, title = "GRAPH1", text.size = 20, return = TRUE, break.nb = 10, classic = TRUE, grid = TRUE) # grouped bars, all the arguments*

*# obs1 <- data.frame(a = 1:1000, group1 = rep(c("G", "H"), times = 500), group2 = rep(LETTERS[1:5], each = 200)) ; fun\_gg\_bar\_mean(data1 = obs1, y = "a", categ = "group1", dot.color = NULL, bar.width = 0.25) # width example. With bar.width = 0.25, three times more space between single bars than the bar width*

*# obs1 <- data.frame(a = 1:1000, group1 = rep(c("G", "H"), times = 500), group2 = rep(LETTERS[1:5], each = 200)) ; fun\_gg\_bar\_mean(data1 = obs1, y = "a", categ = "group1", dot.color = NULL, bar.width = 1) # width example. With bar.width = 1, no space between single bars*

*# obs1 <- data.frame(a = 1:1000, group1 = rep(c("G", "H"), times = 500), group2 = rep(LETTERS[1:5], each = 200)) ; fun\_gg\_bar\_mean(data1 = obs1, y = "a", categ = c("group1", "group2"), dot.color = NULL, bar.width = 0.25) # width example. With bar.width = 0.25, three times more space between sets of grouped bars than the set width*

*# obs1 <- data.frame(a = 1:1000, group1 = rep(c("G", "H"), times = 500), group2 = rep(LETTERS[1:5], each = 200)) ; fun\_gg\_bar\_mean(data1 = obs1, y = "a", categ = c("group1", "group2"), dot.color = NULL, bar.width = 1) # width example. With bar.width = 0, no space between sets of grouped bars*

*# obs1 <- data.frame(a = 1:1000, group1 = rep(c("G", "H"), times = 500), group2 = rep(LETTERS[1:5], each = 200)) ; fun\_gg\_bar\_mean(data1 = obs1, y = "a", categ = c("group1", "group2"), dot.color = NULL, error.bar = "SD", error.bar.width = 1) # width example. With error.bar.width = 1, whiskers have the width of the corresponding bar*

*# obs1 <- data.frame(a = 1:1000, group1 = rep(c("G", "H"), times = 500), group2 = rep(LETTERS[1:5], each = 200)) ; fun\_gg\_bar\_mean(data1 = obs1, y = "a", categ = c("group1", "group2"), dot.color = NULL, error.bar = "SD", error.bar.width = 0) # width example. No whiskers*

*# obs1 <- data.frame(a = 1:1000, group1 = rep(c("G", "H"), times = 500), group2 = rep(LETTERS[1:5], each = 200)) ; fun\_gg\_bar\_mean(data1 = obs1, y = "a", categ = c("group1", "group2"), dot.color = "grey", pt.size = 3, alpha = 1, jitter = 1) # width example. With jitter = 1, dispersion around the corresponding bar width*

*# obs1 <- data.frame(a = 1:1000, group1 = rep(c("G", "H"), times = 500), group2 = rep(LETTERS[1:5], each = 200)) ; fun\_gg\_bar\_mean(data1 = obs1, y = "a", categ = c("group1", "group2"), dot.color = "grey", pt.size = 3, alpha = 1, jitter = 0) # width example. No dispersion*

*# DEBUGGING*

*# data1 <- data.frame(a = 1:20, group1 = rep(c("G", "H"), times = 10)) ; y = names(data1)[1] ; categ = names(data1)[2] ; class.order = list(L1 = NULL) ; legend.name = NULL ; categ.color = c("red", "blue") ; dot.color = "same" ; error.bar = "SEM.TOP" ; bar.width = 0.5 ; error.bar.width = 0.5 ; jitter = 0.25 ; ylim = NULL ; ylog = FALSE ; y.include.zero = FALSE ; top.extra.margin = 0.05 ; bottom.extra.margin = 0 ; xlab = NULL ; ylab = NULL ; pt.size = 3 ; pt.border.size = 0.1 ; alpha = 1 ; show.stat = NULL ; stat.size = 8 ; title = "GRAPH1" ; text.size = 12 ; return = FALSE ; break.nb = NULL ; classic = FALSE ; grid = FALSE ; path.lib = NULL*

*# data1 <- data.frame(a = 1:20, group1 = rep(c("G", "H"), times = 10), group2 = rep(c("A", "B"), each = 10)) ; y = names(data1)[1] ; categ = c(names(data1)[2], names(data1)[3]) ; class.order = list(L1 = NULL, L2 = c("B", "A")) ; legend.name = NULL ; categ.color = c("red", "blue") ; dot.color = "same" ; error.bar = "SEM.TOP" ; bar.width = 0.5 ; error.bar.width = 0.5 ; jitter = 0.25 ; ylim = NULL ; ylog = FALSE ; y.include.zero = FALSE ; top.extra.margin = 0.05 ; bottom.extra.margin = 0 ; xlab = NULL ; ylab = NULL ; pt.size = 3 ; pt.border.size = 0.1 ; alpha = 1 ; show.stat = NULL ; stat.size = 8 ; title = "GRAPH1" ; text.size = 12 ; return = FALSE ; break.nb = NULL ; classic = FALSE ; grid = FALSE ; path.lib = NULL*

*# data1 <- data.frame(a = 1:20, group1 = rep(c("G", "H"), times = 10), group2 = rep(c("A", "B"), each = 10)) ; y = names(data1)[1] ; categ = c(names(data1)[2], names(data1)[3]) ; class.order = list(L1 = NULL, L2 = c("B", "A")) ; legend.name = NULL ; categ.color = NULL ; dot.color = "same" ; error.bar = NULL ; bar.width = 0.5 ; error.bar.width = 0.5 ; jitter = 0.25 ; ylim = NULL ; ylog = TRUE ; y.include.zero = FALSE ; top.extra.margin = 0.05 ; bottom.extra.margin = 0 ; xlab = NULL ; ylab = NULL ; pt.size = 3 ; pt.border.size = 0.1 ; alpha = 0.5 ; show.stat = NULL ; stat.size = 8 ; title = "" ; text.size = 12 ; return = FALSE ; break.nb = NULL ; classic = FALSE ; grid = FALSE ; path.lib = NULL*

*# data1 <- data.frame(a = 1:20, group1 = rep(c("G", "H"), times = 10), group2 = rep(c("A", "B"), each = 10), bar.color = rep(c("brown", "orange"), each = 10)) ; y = names(data1)[1] ; categ = c(names(data1)[2], names(data1)[3]) ; class.order = list(L1 = NULL, L2 = c("B", "A")) ; legend.name = NULL ; categ.color = data1$bar.color ; dot.color = "same" ; error.bar = "SD" ; bar.width = 0.5 ; error.bar.width = 0.5 ; jitter = 0.25 ; ylim = NULL ; ylog = TRUE ; y.include.zero = FALSE ; top.extra.margin = 0.05 ; bottom.extra.margin = 0 ; xlab = NULL ; ylab = NULL ; pt.size = 3 ; pt.border.size = 0.1 ; alpha = 0.5 ; show.stat = NULL ; stat.size = 8 ; title = "" ; text.size = 12 ; return = FALSE ; break.nb = NULL ; classic = FALSE ; grid = FALSE ; path.lib = NULL*

*# data1 <- data.frame(a = 1:20, group1 = rep(c("G", "H"), times = 10), group2 = rep(c("A", "B"), each = 10), bar.color = rep(c("brown", "orange"), each = 10)) ; data1[2:3, 1] <- NA ; data1[7:8, 2] <- NA ; y = names(data1)[1] ; categ = c(names(data1)[2], names(data1)[3]) ; class.order = list(L1 = NULL, L2 = c("B", "A")) ; legend.name = NULL ; categ.color = na.omit(data1)$bar.color ; dot.color = "same" ; error.bar = "SD" ; bar.width = 0.5 ; error.bar.width = 0.5 ; jitter = 0.25 ; ylim = NULL ; ylog = TRUE ; y.include.zero = FALSE ; top.extra.margin = 0.05 ; bottom.extra.margin = 0 ; xlab = NULL ; ylab = NULL ; pt.size = 3 ; pt.border.size = 0.1 ; alpha = 0.5 ; show.stat = "above" ; stat.size = 4 ; title = "" ; text.size = 12 ; return = FALSE ; break.nb = NULL ; classic = FALSE ; grid = FALSE ; path.lib = NULL*

*# function name*

**}**

######## fun\_gg\_bar\_prop() #### ggplot2 proportion barplot

######## fun\_gg\_strip() #### ggplot2 stripchart + mean/median

######## fun\_gg\_violin() #### ggplot2 violins

######## fun\_gg\_line() #### ggplot2 lines + background dots and error bars

######## 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 ?**

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

**fun\_gg\_heatmap <- function(data1, legend.name = "", low.color = "blue", high.color = "red", mid.color = "white", limit = NULL, midpoint = NULL, title = "", text.size = 12, show.scale = TRUE, data2 = NULL, color2 = "black", alpha2 = 0.5, invert2 = FALSE, return = FALSE, path.lib = 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.name: character string of the heatmap scale legend*

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

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

*# mid.color: same as low.color but for the middle scale value*

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

*# midpoint: single numeric value defining the middle color scale values. If NULL, take the mean of data1 values*

*# title: character string of the graph title*

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

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

*# 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)?*

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

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

*# REQUIRED PACKAGES*

*# ggplot2*

*# reshape2*

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

*# fun\_param\_check()*

*# fun\_pack\_import()*

*# RETURN*

*# a heatmap*

*# the graph info if return argument is TRUE*

*# 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.name = "VALUE", title = "GRAPH 1", text.size = 5, data2 = matrix(rep(c(1,0,0,0), 4), ncol = 4), invert2 = FALSE, return = TRUE)*

*# 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)))*

*# DEBUGGING*

*# data1 = matrix(1:16, ncol = 4) ; legend.name = "" ; low.color = "blue" ; high.color = "red" ; mid.color = "white" ; limit = range(data1, na.rm = TRUE) ; midpoint = mean(data1, na.rm = TRUE) ; title = "GRAPH 1" ; text.size = 12 ; show.scale = TRUE ; data2 = NULL ; color2 = "black" ; alpha2 = 0.5 ; invert2 = FALSE ; return = FALSE ; path.lib = NULL*

*# data1 = matrix(1:16, ncol = 4) ; legend.name = "" ; low.color = "blue" ; high.color = "red" ; mid.color = "white" ; limit = range(data1, na.rm = TRUE) ; midpoint = mean(data1, na.rm = TRUE) ; title = "GRAPH 1" ; text.size = 12 ; show.scale = TRUE ; data2 = matrix(rep(c(1,0,0,0), 4), ncol = 4) ; color2 = "black" ; alpha2 = 0.5 ; invert2 = FALSE ; return = TRUE ; path.lib = NULL*

*# function name*

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

*# end function name*

*# required function checking*

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

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

**stop(tempo.cat)**

**}**

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

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

**stop(tempo.cat)**

**}**

*# end required function checking*

*# argument checking*

**arg.check <- NULL** *# for function debbuging*

**checked.arg.names <- NULL** *# for function debbuging*

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

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

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

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

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

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

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

**}**

**}else{**

**tempo.cat <- paste0("\n\n================\n\nERROR 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")**

**stop(tempo.cat)**

**}**

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

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

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

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

**stop(tempo.cat)**

**}**

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

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

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

**stop(tempo.cat)**

**}**

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

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

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

**stop(tempo.cat)**

**}**

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

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

**}**

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

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

**}**

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

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

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

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

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

**tempo <- fun\_param\_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("\n\n================\n\nERROR IN ", function.name, ": MATRIX IN data2 MUST BE MADE OF 0 AND 1 ONLY (MASK MATRIX)\n\n================\n\n")**

**stop(tempo.cat)**

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

**tempo.cat <- paste0("\n\n================\n\nERROR 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 = " "), "\n\n================\n\n")**

**stop(tempo.cat)**

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

**tempo.cat <- paste0("\n\n================\n\nERROR 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 = " "), "\n\n================\n\n")**

**stop(tempo.cat)**

**}**

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

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

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

**tempo <- fun\_param\_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("\n\n================\n\nERROR 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")**

**stop(tempo.cat)**

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

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

**stop(tempo.cat)**

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

**tempo.cat <- paste0("\n\n================\n\nERROR 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 = " "), "\n\n================\n\n")**

**stop(tempo.cat)**

**}**

**}else{**

**tempo.cat <- paste0("\n\n================\n\nERROR 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")**

**stop(tempo.cat)**

**}**

**}**

**tempo <- fun\_param\_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 mid.color start by #*

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

**stop(tempo.cat)**

**}**

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

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

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

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

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

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

**cat(paste0("\n\n============\n\nERROR IN ", function.name, ": \nDIRECTORY PATH INDICATED IN THE path.lib PARAMETER DOES NOT EXISTS: ", path.lib, "\n\n============\n\n"))**

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

**}**

**}**

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

**stop()** *# nothing else because print = TRUE by default in fun\_param\_check()*

**}**

*# 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\_param\_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\_param\_check()*

*# end argument checking*

*# package checking*

**fun\_pack\_import(req.package = c("reshape2", "ggplot2"), path.lib = path.lib)**

*# 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(is.null(limit)){**

**limit <- range(data1[, 3], na.rm = TRUE)**

**}**

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

**midpoint <- mean(data1[, 3], na.rm = TRUE)**

**}**

**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(invert2 == FALSE){**

**data2[data2[, 3] == 1, 3] <- color2**

**data2[data2[, 3] == 0, 3] <- NA**

**}else{**

**data2[data2[, 3] == 0, 3] <- color2**

**data2[data2[, 3] == 1, 3] <- NA**

**}**

**}**

**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(data = data1, mapping = ggplot2::aes\_string(x = names(data1)[2], y = names(data1)[1], fill = names(data1)[3])))**

**assign(paste0(tempo.gg.name, tempo.gg.count <- tempo.gg.count + 1), ggplot2::geom\_raster(show.legend = show.scale))**

**assign(paste0(tempo.gg.name, tempo.gg.count <- tempo.gg.count + 1), ggplot2::scale\_fill\_gradient2(low = low.color, high = high.color, mid = mid.color, midpoint = midpoint, limit = limit, breaks = c(limit, midpoint), name = legend.name))**

**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)[2], y = names(data2)[1], group = names(data2)[3]), fill = data2[, 3], alpha = alpha2, show.legend = FALSE))**

**}**

**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))**

**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(**

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

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

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

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

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

**))**

**suppressWarnings(print(eval(parse(text = paste(paste0(tempo.gg.name, 1:tempo.gg.count), collapse = " + ")))))**

**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(output)**

**}**

**}**

######## fun\_gg\_empty\_graph() #### text to display for empty graphs

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

**fun\_gg\_empty\_graph <- function(text, text.size = 12, title = NULL, path.lib = NULL){**

*# AIM*

*# display an empty ggplot2 plot with a text in the middle of the window (for instance to specify that no plot can be drawn)*

*# ARGUMENTS*

*# text: character string of the message to display*

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

*# title: character string of the graph title*

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

*# REQUIRED PACKAGES*

*# ggplot2*

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

*# fun\_param\_check()*

*# fun\_pack\_import()*

*# RETURN*

*# an empty plot*

*# EXAMPLES*

*# fun\_gg\_empty\_graph(text = "NO GRAPH", title = "GRAPH1")*

*# DEBUGGING*

*# text = "NO GRAPH" ; text.size = 12 ; title = "GRAPH1" ; path.lib = NULL*

*# function name*

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

*# end function name*

*# required function checking*

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

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

**stop(tempo.cat)**

**}**

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

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

**stop(tempo.cat)**

**}**

*# end required function checking*

*# argument checking*

**arg.check <- NULL** *# for function debbuging*

**checked.arg.names <- NULL** *# for function debbuging*

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

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

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

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

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

**}**

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

**stop()** *# nothing else because print = TRUE by default in fun\_param\_check()*

**}**

*# 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\_param\_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\_param\_check()*

*# end argument checking*

*# package checking*

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

*# end package checking*

*# main code*

**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::geom\_text(data = data.frame(x = 1, y = 1), aes(x = x, y = y, label = text)))**

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

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

**suppressWarnings(print(eval(parse(text = paste(paste0(tempo.gg.name, 1:tempo.gg.count), collapse = " + ")))))**

**}**

################ Graphic extraction

######## fun\_var\_trim\_display() #### Display values from a quantitative variable and trim according to defined cut-offs

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

**fun\_var\_trim\_display <- function(data, displayed.nb = NULL, single.value.display = FALSE, trim.method = "", trim.cutoffs = c(0.05, 0.975), interval.scale.disp = TRUE, down.space = 0.75, left.space = 0.75, up.space = 0.3, right.space = 0.25, orient = 1, dist.legend = 0.37, box.type = "l", amplif.label = 1.25, amplif.axis = 1.25, std.x.range = TRUE, std.y.range = TRUE, cex.pt = 0.2, col.box = hsv(0.55, 0.8, 0.8), x.nb.inter.tick = 4, y.nb.inter.tick = 0, tick.length = 1, sec.tick.length = 0.75, corner.text = "", amplif.legend = 1, magnific.corner.text = 0.75, trim.return = FALSE){**

*# AIM*

*# trim and display values from a numeric vector or matrix*

*# plot 4 graphs: stripchart of values, stripchart of rank of values, hitogramme and normal QQPlot*

*# different kinds of intervals are displayed on the top of graphes to facilitate the analysis of the variable and a trimming setting*

*# the trimming interval chosen is displayed on top of graphs*

*# both trimmed and not trimmed values are returned in a list*

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

*# fun\_param\_check()*

*# ARGUMENTS*

*# data: values to plot (either a numeric vector or a numeric matrix)*

*# displayed.nb: number of values displayed. If NULL, all the values are displayed. Otherwise, if the number of values is over displayed.nb, then displayed.nb values are displayed after random selection*

*# single.value.display: provide the 4 graphs if data is made of a single (potentially repeated value)? If FALSE, an empty graph is displayed if data is made of a single (potentially repeated value). And the return list is made of NULL compartments*

*# trim.method: Write "" if not required. write "mean.sd" if mean +/- sd has to be displayed as a trimming interval (only recommanded for normal distribution). Write "quantile" to display a trimming interval based on quantile cut-offs. No other possibility allowed. See trim.cutoffs below*

*# trim.cutoffs: 2 values cutoff for the trimming interval displayed, each value between 0 and 1. Not used if trim.method == "".The couple of values c(lower, upper) represents the lower and upper boundaries of the trimming interval (in proportion), which represent the interval of distribution kept (between 0 and 1). Example: trim.cutoffs = c(0.05, 0.975). What is strictly kept for the display is ]lower , upper[, boundaries excluded. Using the "mean.sd" method, 0.025 and 0.975 represent 95% CI which is mean +/- 1.96 \* sd*

*# interval.scale.disp: display sd and quantiles intervals on top of graphs ?*

*# down.space: lower vertical margin (in inches, mai argument of par())*

*# left.space: left horizontal margin (in inches, mai argument of par())*

*# up.space: upper vertical margin between plot region and grapical window (in inches, mai argument of par())*

*# right.space: right horizontal margin (in inches, mai argument of par())*

*# orient: scale number orientation (las argument of par()). 0, always parallel to the axis; 1, always horizontal; 2, always perpendicular to the axis; 3, always vertical*

*# dist.legend: numeric value that moves axis legends away in inches (first number of mgp argument of par() but in inches thus / 0.2)*

*# box.type: bty argument of par(). Either "o", "l", "7", "c", "u", "]", the resulting box resembles the corresponding upper case letter. A value of "n" suppresses the box*

*# amplif.label: increase or decrease the size of the text in legends*

*# amplif.axis: increase or decrease the size of the scale numbers in axis*

*# std.x.range: standard range on the x-axis? TRUE (no range extend) or FALSE (4% range extend). Controls xaxs argument of par() (TRUE is xaxs = "i", FALSE is xaxs = "r")*

*# std.y.range: standard range on the y-axis? TRUE (no range extend) or FALSE (4% range extend). Controls yaxs argument of par() (TRUE is yaxs = "i", FALSE is yaxs = "r")*

*# cex.pt: size of points in stripcharts (in inches, thus cex.pt will be thereafter / 0.2)*

*# col.box: color of boxplot*

*# x.nb.inter.tick: number of secondary ticks between main ticks on x-axis (only if not log scale). Zero means non secondary ticks*

*# y.nb.inter.tick: number of secondary ticks between main ticks on y-axis (only if not log scale). Zero means non secondary ticks*

*# tick.length: length of the ticks (1 means complete the distance between the plot region and the axis numbers, 0.5 means half the length, etc. 0 means no tick*

*# sec.tick.length: length of the secondary ticks (1 means complete the distance between the plot region and the axis numbers, 0.5 means half the length, etc., 0 for no ticks)*

*# corner.text: text to add at the top right corner of the window*

*# amplif.legend: increase or decrease the size of the text of legend*

*# magnific.corner.text: increase or decrease the size of the text*

*# trim.return: return the trimmed and non trimmed values? NULL returned for trimmed and non trimmed values if trim.method == ""*

*# RETURN*

*# a list containing:*

*# $trim.method: correspond to trim.method above*

*# $trim.cutoffs: correspond to trim.cutoffs above*

*# $real.trim.cutoffs: the two boundary values (in the unit of the numeric vector or numeric matrix analyzed). NULL*

*# $trimmed.values: the values outside of the trimming interval as defined in trim.cutoffs above*

*# $kept.values: the values inside the trimming interval as defined in trim.cutoffs above*

*# EXAMPLES*

*# fun\_var\_trim\_display(data = c(1:100, 1:10), displayed.nb = NULL, single.value.display = FALSE, trim.method = "mean.sd", trim.cutoffs = c(0.05, 0.975), interval.scale.disp = TRUE, down.space = 0.75, left.space = 0.75, up.space = 0.3, right.space = 0.25, orient = 1, dist.legend = 0.37, box.type = "l", amplif.label = 1.25, amplif.axis = 1.25, std.x.range = TRUE, std.y.range = TRUE, cex.pt = 0.2, col.box = hsv(0.55, 0.8, 0.8), x.nb.inter.tick = 4, y.nb.inter.tick = 0, tick.length = 0.5, sec.tick.length = 0.3, corner.text = "", amplif.legend = 1, magnific.corner.text = 0.75, trim.return = TRUE)*

*# DEBUGGING*

*# data = c(1:100, 1:10) ; displayed.nb = NULL ; single.value.display = FALSE ; trim.method = "quantile" ; trim.cutoffs = c(0.05, 0.975) ; interval.scale.disp = TRUE ; down.space = 1 ; left.space = 1 ; up.space = 0.5 ; right.space = 0.25 ; orient = 1 ; dist.legend = 0.5 ; box.type = "l" ; amplif.label = 1 ; amplif.axis = 1 ; std.x.range = TRUE ; std.y.range = TRUE ; cex.pt = 0.1 ; col.box = hsv(0.55, 0.8, 0.8) ; x.nb.inter.tick = 4 ; y.nb.inter.tick = 0 ; tick.length = 0.5 ; sec.tick.length = 0.3 ; corner.text = "" ; amplif.legend = 1 ; magnific.corner.text = 0.75 ; trim.return = TRUE # for function debugging*

*# function name*

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

*# end function name*

*# required function checking*

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

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

**stop(tempo.cat)**

**}**

*# end required function checking*

*# argument checking*

*# argument checking without fun\_param\_check()*

**if( ! (all(class(data) == "numeric") | all(class(data) == "integer") | (all(class(data) == "matrix") & mode(data) == "numeric"))){**

**tempo.cat <- paste0("\n\n================\n\nERROR IN ", function.name, ": data ARGUMENT MUST BE A NUMERIC VECTOR OR NUMERIC MATRIX\n\n================\n\n")**

**stop(tempo.cat)**

**}**

*# end argument checking without fun\_param\_check()*

*# argument checking with fun\_param\_check()*

**arg.check <- NULL** *# for function debbuging*

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

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

**if( ! is.null(displayed.nb)){**

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

**if(displayed.nb < 2){**

**tempo.cat <- paste0("\n\n================\n\nERROR IN ", function.name, ": displayed.nb ARGUMENT MUST BE A SINGLE INTEGER VALUE GREATER THAN 1 AND NOT: ", paste(displayed.nb, collapse = " "), "\n\n================\n\n")**

**stop(tempo.cat)**

**}**

**}**

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

**tempo <- fun\_param\_check(data = trim.method, options = c("", "mean.sd", "quantile"), length = 1, fun.name = function.name) ; eval(ee)**

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

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

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

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

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

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

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

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

**tempo <- fun\_param\_check(data = box.type, options = c("o", "l", "7", "c", "u", "]", "n"), length = 1, fun.name = function.name) ; eval(ee)**

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

**stop()** *# nothing else because print = TRUE by default in fun\_param\_check()*

**}**

*# end argument checking with fun\_param\_check()*

*# 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\_param\_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\_param\_check()*

*# end argument checking*

*# main code*

**if(class(data) == "matrix"){**

**data <- as.vector(data)**

**}**

**color.cut <- hsv(0.75, 1, 1)** *# color of interval selected*

**col.mean <- hsv(0.25, 1, 0.8)** *# color of interval using mean+/-sd*

**col.quantile <- "orange"** *# color of interval using quantiles*

**quantiles.selection <- c(0.01, 0.025, 0.05, 0.1, 0.25, 0.5, 0.75, 0.9, 0.95, 0.975, 0.99)** *# quantiles used in axis to help for choosing trimming cutoffs*

**if(single.value.display == FALSE & length(unique(data)) == 1){**

**par(bty = "n", xaxt = "n", yaxt = "n", xpd = TRUE)**

**plot(1, pch = 16, col = "white", xlab = "", ylab = "")**

**text(x = 1, y = 1, paste0("No graphic displayed\nBecause data made of a single different value (", formatC(as.double(table(data))), ")"), cex = 2)**

**output <- list(trim.method = NULL, trim.cutoffs = NULL, real.trim.cutoffs = NULL, trimmed.values = NULL, kept.values = NULL)**

**}else{**

**output <- list(trim.method = trim.method, trim.cutoffs = trim.cutoffs, real.trim.cutoffs = NULL, trimmed.values = NULL, kept.values = NULL)**

**fun.rug <- function(sec.tick.length.f = sec.tick.length, x.nb.inter.tick.f = x.nb.inter.tick, y.nb.inter.tick.f = y.nb.inter.tick){**

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

**inter.tick.unit <- (par("xaxp")[2] - par("xaxp")[1]) / par("xaxp")[3]**

**par.ini <- par()[c("xpd", "tcl")]**

**par(xpd = FALSE)**

**par(tcl = -par()$mgp[2] \* sec.tick.length.f)** *# tcl gives the length of the ticks as proportion of line text, knowing that mgp is in text lines. So the main ticks are a 0.5 of the distance of the axis numbers by default. The sign provides the side of the tick (negative for outside of the plot region)*

**suppressWarnings(rug(seq(par("xaxp")[1] - 10 \* inter.tick.unit, par("xaxp")[2] + 10 \* inter.tick.unit, by = inter.tick.unit / (1 + x.nb.inter.tick.f)), ticksize = NA, side = 1))** *# ticksize = NA to allow the use of par()$tcl value*

**par(par.ini)**

**rm(par.ini)**

**}**

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

**inter.tick.unit <- (par("yaxp")[2] - par("yaxp")[1]) / par("yaxp")[3]**

**par.ini <- par()[c("xpd", "tcl")]**

**par(xpd = FALSE)**

**par(tcl = -par()$mgp[2] \* sec.tick.length.f)** *# tcl gives the length of the ticks as proportion of line text, knowing that mgp is in text lines. So the main ticks are a 0.5 of the distance of the axis numbers by default. The sign provides the side of the tick (negative for outside of the plot region)*

**suppressWarnings(rug(seq(par("yaxp")[1] - 10 \* inter.tick.unit, par("yaxp")[2] + 10 \* inter.tick.unit, by = inter.tick.unit / (1 + y.nb.inter.tick.f)), ticksize = NA, side = 2))** *# ticksize = NA to allow the use of par()$tcl value*

**par(par.ini)**

**rm(par.ini)**

**}**

**}**

**fun.add.cut <- function(data.f, trim.method.f = trim.method, trim.cutoffs.f = trim.cutoffs, color.cut.f = color.cut, return.f = FALSE){**

*# DEBUGGING*

*# data.f = data ; trim.method.f = "mean.sd"; trim.cutoffs.f = trim.cutoffs ; color.cut.f = color.cut ; return.f = TRUE*

**real.trim.cutoffs.f <- NULL**

**if(trim.method.f != ""){**

**data.f <- sort(data.f)**

**par.ini <- par()$xpd**

**par(xpd = FALSE)**

**if(trim.method.f == "mean.sd"){**

**real.trim.cutoffs.f <- qnorm(trim.cutoffs.f, mean(data.f, na.rm = TRUE), sd(data.f, na.rm = TRUE))**

**abline(v = qnorm(trim.cutoffs.f, mean(data.f, na.rm = TRUE), sd(data.f, na.rm = TRUE)), col = color.cut.f)**

**segments(qnorm(trim.cutoffs.f[1], mean(data.f, na.rm = TRUE), sd(data.f, na.rm = TRUE)), par()$usr[4] \* 0.75, qnorm(trim.cutoffs.f[2], mean(data.f, na.rm = TRUE), sd(data.f, na.rm = TRUE)), par()$usr[4] \* 0.75, col = color.cut.f)**

**}**

**if(trim.method.f == "quantile"){**

**real.trim.cutoffs.f <- quantile(data.f, probs = trim.cutoffs.f, type = 7)**

**abline(v = quantile(data.f, probs = trim.cutoffs.f, type = 7), col = color.cut.f)**

**segments(quantile(data.f, probs = trim.cutoffs.f[1], type = 7), par()$usr[4] \* 0.75, quantile(data.f, probs = trim.cutoffs.f[2], type = 7), par()$usr[4] \* 0.75, col = color.cut.f)**

**}**

**par(par.ini)**

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

**trimmed.values.f <- data.f[data.f <= real.trim.cutoffs.f[1] | data.f >= real.trim.cutoffs.f[2]]**

**kept.values.f <- data.f[data.f > real.trim.cutoffs.f[1] & data.f < real.trim.cutoffs.f[2]]**

**}**

**}else{**

**real.trim.cutoffs.f <- NULL**

**trimmed.values.f <- NULL**

**kept.values.f <- NULL**

**}**

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

**output <- list(trim.method = trim.method.f, trim.cutoffs = trim.cutoffs.f, real.trim.cutoffs = real.trim.cutoffs.f, trimmed.values = trimmed.values.f, kept.values = kept.values.f)**

**return(output)**

**}**

**}**

**fun.interval.scale.display <- function(data.f, col.quantile.f = col.quantile, quantiles.selection.f = quantiles.selection, col.mean.f = col.mean){** *# intervals on top of graphs*

**par.ini <- par()[c("mgp", "xpd")]**

**par(mgp = c(0.25, 0.25, 0), xpd = NA)**

**axis(side = 3, at = c(par()$usr[1], par()$usr[2]), labels = rep("", 2), col = col.quantile.f, lwd.ticks = 0)**

**par(xpd = FALSE)**

**axis(side = 3, at = quantile(as.vector(data.f), probs = quantiles.selection.f, type = 7), labels = quantiles.selection.f, col.axis = col.quantile.f, col = col.quantile.f)**

**par(mgp = c(1.75, 1.75, 1.5), xpd = NA)**

**axis(side = 3, at = c(par()$usr[1], par()$usr[2]), labels = rep("", 2), col = col.mean.f, lwd.ticks = 0)**

**par(xpd = FALSE)**

**axis(side = 3, at = m + s \* qnorm(quantiles.selection.f), labels = formatC(round(qnorm(quantiles.selection.f), 2)), col.axis = col.mean.f, col = col.mean.f, lwd.ticks = 1)**

**par(par.ini)**

**}**

**zone<-matrix(1:4, ncol=2)**

**layout(zone)**

**par(omi = c(0, 0, 1.5, 0), mai = c(down.space, left.space, up.space, right.space), las = orient, mgp = c(dist.legend / 0.2, 0.5, 0), xpd = FALSE, bty= box.type, cex.lab = amplif.label, cex.axis = amplif.axis, xaxs = ifelse(std.x.range, "i", "r"), yaxs = ifelse(std.y.range, "i", "r"))**

**par(tcl = -par()$mgp[2] \* tick.length)** *# tcl gives the length of the ticks as proportion of line text, knowing that mgp is in text lines. So the main ticks are a 0.5 of the distance of the axis numbers by default. The sign provides the side of the tick (negative for outside of the plot region)*

**if(is.null(displayed.nb)){**

**sampled.data <- as.vector(data)**

**if(corner.text == ""){**

**corner.text <- paste0("ALL VALUES OF THE DATASET DISPLAYED")**

**}else{**

**corner.text <- paste0(corner.text, "\nALL VALUES OF THE DATASET DISPLAYED")**

**}**

**}else{**

**if(length(as.vector(data)) > displayed.nb){**

**sampled.data <- sample(as.vector(data), displayed.nb, replace = FALSE)**

**if(corner.text == ""){**

**corner.text <- paste0("BEWARE: ONLY ", displayed.nb, " VALUES ARE DISPLAYED AMONG THE ", length(as.vector(data)), " VALUES OF THE DATASET ANALYZED")**

**}else{**

**corner.text <- paste0(corner.text, "\nBEWARE: ONLY ", displayed.nb, " VALUES ARE DISPLAYED AMONG THE ", length(as.vector(data)), " VALUES OF THE DATASET ANALYZED")**

**}**

**}else{**

**sampled.data <- as.vector(data)**

**if(corner.text == ""){**

**corner.text <- paste0("BEWARE: THE DISPLAYED NUMBER OF VALUES PARAMETER ", deparse(substitute(displayed.nb)), " HAS BEEN SET TO ", displayed.nb, " WHICH IS ABOVE THE NUMBER OF VALUES OF THE DATASET ANALYZED -> ALL VALUES DISPLAYED")**

**}else{**

**corner.text <- paste0(corner.text, "\nBEWARE: THE DISPLAYED NUMBER OF VALUES PARAMETER ", deparse(substitute(displayed.nb)), " HAS BEEN SET TO ", displayed.nb, " WHICH IS ABOVE THE NUMBER OF VALUES OF THE DATASET ANALYZED -> ALL VALUES DISPLAYED")**

**}**

**}**

**}**

**stripchart(sampled.data, method="jitter", jitter=0.4, vertical=FALSE, ylim=c(0.5, 1.5), group.names = "", xlab = "Value", ylab="", pch=1, cex = cex.pt / 0.2)**

**fun.rug(y.nb.inter.tick.f = 0)**

**boxplot(as.vector(data), horizontal=TRUE, add=TRUE, boxwex = 0.4, staplecol = col.box, whiskcol = col.box, medcol = col.box, boxcol = col.box, range = 0, whisklty = 1)**

**m <- mean(as.vector(data), na.rm = TRUE)**

**s <- sd(as.vector(data), na.rm = TRUE)**

**segments(m, 0.8, m, 1, lwd=2, col="red")** *# mean*

**segments(m -1.96 \* s, 0.9, m + 1.96 \* s, 0.9, lwd=1, col="red")** *# mean*

**graph.xlim <- par()$usr[1:2]** *# for hist() and qqnorm() below*

**if(interval.scale.disp == TRUE){**

**fun.interval.scale.display(data.f = data)**

**if(corner.text == ""){**

**corner.text <- paste0("MULTIPLYING FACTOR DISPLAYED (MEAN +/- SD) ON SCALES: ", paste(formatC(round(qnorm(quantiles.selection), 2))[-(1:(length(quantiles.selection) - 1) / 2)], collapse = ", "), "\nQUANTILES DISPLAYED ON SCALES: ", paste(quantiles.selection, collapse = ", "))**

**}else{**

**corner.text <- paste0(corner.text, "\nMULTIPLYING FACTOR DISPLAYED (MEAN +/- SD) ON SCALES: ", paste(formatC(round(qnorm(quantiles.selection), 2))[-(1:(length(quantiles.selection) - 1) / 2)], collapse = ", "), "\nQUANTILES DISPLAYED ON SCALES: ", paste(quantiles.selection, collapse = ", "))**

**}**

**}**

**output.tempo <- fun.add.cut(data.f = data, return.f = TRUE)** *# to recover real.trim.cutoffs*

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

**output <- output.tempo**

**}**

**par(xpd = NA)**

**if(trim.method != ""){**

**if(corner.text == ""){**

**corner.text <- paste0("SELECTED CUT-OFFS (PROPORTION): ", paste(trim.cutoffs, collapse = ", "), "\nSELECTED CUT-OFFS: ", paste(output.tempo$real.trim.cutoffs, collapse = ", "))**

**}else{**

**corner.text <- paste0(corner.text, "\nSELECTED CUT-OFFS (PROPORTION): ", paste(trim.cutoffs, collapse = ", "), "\nSELECTED CUT-OFFS: ", paste(output.tempo$real.trim.cutoffs, collapse = ", "))**

**}**

**if(interval.scale.disp == TRUE){**

**legend(x = (par("usr")[1] - ((par("usr")[2] - par("usr")[1]) / (par("plt")[2] - par("plt")[1])) \* par("plt")[1] - ((par("usr")[2] - par("usr")[1]) / (par("omd")[2] - par("omd")[1])) \* par("omd")[1]), y = (par("usr")[4] + ((par("usr")[4] - par("usr")[3]) / (par("plt")[4] - par("plt")[3])) \* (1 - par("plt")[4]) + ((par("usr")[4] - par("usr")[3]) / (par("omd")[4] - par("omd")[3])) \* (1 - par("omd")[4]) / 2), legend = c(c("min, Q1, Median, Q3, max"), "mean +/- 1.96sd", paste0("Trimming interval: ", paste0(trim.cutoffs, collapse = " , ")), "Mean +/- sd multiplying factor", "Quantile"), yjust = 0, lty=1, col=c(col.box, "red", color.cut, col.mean, col.quantile), bty="n", cex = amplif.legend)**

**}else{**

**legend(x = (par("usr")[1] - ((par("usr")[2] - par("usr")[1]) / (par("plt")[2] - par("plt")[1])) \* par("plt")[1] - ((par("usr")[2] - par("usr")[1]) / (par("omd")[2] - par("omd")[1])) \* par("omd")[1]), y = (par("usr")[4] + ((par("usr")[4] - par("usr")[3]) / (par("plt")[4] - par("plt")[3])) \* (1 - par("plt")[4]) + ((par("usr")[4] - par("usr")[3]) / (par("omd")[4] - par("omd")[3])) \* (1 - par("omd")[4]) / 2), legend = c(c("min, Q1, Median, Q3, max"), "mean +/- 1.96sd", paste0("Trimming interval: ", paste0(trim.cutoffs, collapse = " , "))), yjust = 0, lty=1, col=c(col.box, "red", color.cut), bty="n", cex = amplif.legend, y.intersp=1.25)**

**}**

**}else{**

**if(interval.scale.disp == TRUE){**

**legend(x = (par("usr")[1] - ((par("usr")[2] - par("usr")[1]) / (par("plt")[2] - par("plt")[1])) \* par("plt")[1] - ((par("usr")[2] - par("usr")[1]) / (par("omd")[2] - par("omd")[1])) \* par("omd")[1]), y = (par("usr")[4] + ((par("usr")[4] - par("usr")[3]) / (par("plt")[4] - par("plt")[3])) \* (1 - par("plt")[4]) + ((par("usr")[4] - par("usr")[3]) / (par("omd")[4] - par("omd")[3])) \* (1 - par("omd")[4]) / 2), legend = c(c("min, Q1, Median, Q3, max"), "mean +/- sd", "Mean +/- sd multiplying factor", "Quantile"), yjust = 0, lty=1, col=c(col.box, "red", col.mean, col.quantile), bty="n", cex = amplif.legend)**

**}else{**

**legend(x = (par("usr")[1] - ((par("usr")[2] - par("usr")[1]) / (par("plt")[2] - par("plt")[1])) \* par("plt")[1] - ((par("usr")[2] - par("usr")[1]) / (par("omd")[2] - par("omd")[1])) \* par("omd")[1]), y = (par("usr")[4] + ((par("usr")[4] - par("usr")[3]) / (par("plt")[4] - par("plt")[3])) \* (1 - par("plt")[4]) + ((par("usr")[4] - par("usr")[3]) / (par("omd")[4] - par("omd")[3])) \* (1 - par("omd")[4]) / 2), legend = c(c("min, Q1, Median, Q3, max"), "mean +/- sd"), yjust = 0, lty=1, col=c(col.box, "red"), bty="n", cex = amplif.legend, y.intersp=1.25)**

**}**

**}**

**par(xpd = FALSE, xaxs = ifelse(std.x.range, "i", "r"), yaxs = ifelse(std.y.range, "i", "r"))**

**hist(as.vector(data), main = "", breaks = seq(min(as.vector(data), na.rm = TRUE), max(as.vector(data), na.rm = TRUE), length.out = length(as.vector(data)) / 10), xlim = graph.xlim, xlab = "Value", ylab="Density", col = grey(0.25))**

**abline(h = par()$usr[3])**

**fun.rug()**

**if(interval.scale.disp == TRUE){**

**fun.interval.scale.display(data.f = data)**

**}**

**fun.add.cut(data.f = data)**

**par(xaxs = ifelse(std.x.range, "i", "r"))**

**stripchart(rank(sampled.data), method="stack", vertical=FALSE, ylim=c(0.99, 1.3), group.names = "", xlab = "Rank of values", ylab="", pch=1, cex = cex.pt / 0.2)**

**fun.rug(y.nb.inter.tick.f = 0)**

**x.text <- par("usr")[2] + (par("usr")[2] - par("usr")[1]) / (par("plt")[2] - par("plt")[1]) \* (1 - par("plt")[2]) / 2**

**y.text <- (par("usr")[4] + ((par("usr")[4] - par("usr")[3]) / (par("plt")[4] - par("plt")[3])) \* (1 - par("plt")[4]) + ((par("usr")[4] - par("usr")[3]) / ((par()$omd[4] / 2) \* ((par("plt")[4] - par("plt")[3])))) \* (1 - par("omd")[4]))** *# BEWARE. Here in "(par()$omd[4] / 2", division by two because there are 2 graphs staked on the y axis, and not one*

**par(xpd=NA)**

**text(x = x.text, y = y.text, paste0(corner.text), adj=c(1, 1.1), cex = magnific.corner.text)** *# text at the topright corner*

**par(xpd=FALSE)**

**par(xaxs = ifelse(std.x.range, "i", "r"), yaxs = ifelse(std.y.range, "i", "r"))**

**qqnorm(as.vector(sampled.data), main = "", datax = TRUE, ylab = "Value", pch = 1, col = "red", cex = cex.pt / 0.2)**

**fun.rug()**

**if(diff(quantile(as.vector(data), probs = c(0.25, 0.75), na.rm = TRUE)) != 0){** *# otherwise, error generated*

**qqline(as.vector(data), datax = TRUE)**

**}**

**if(interval.scale.disp == TRUE){**

**fun.interval.scale.display(data.f = data)**

**}**

**fun.add.cut(data.f = data)**

**}**

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

**return(output)**

**}**

**}**

######## fun\_segmentation() #### Segment a dot cloud on a scatterplot and define the dots from another cloud outside the segmentation

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

**fun\_segmentation <- function(data1, x1, y1, x.range.split = NULL, x.step.factor = 10, y.range.split = NULL, y.step.factor = 10, error = 0, data2 = NULL, x2, y2, xy.cross.kind = "&", graph.check = FALSE, graph.path = "C:/Users/Gael/Desktop/", path.lib = NULL){**

*# AIM*

*# If data1 is a data frame corresponding to the data set of a scatterplot (with a x column for x-axis values and a y column for the y-axis column), then fun\_segmentation() delimits a frame around the dots cloud using a sliding window set by x.range.split and x.step.factor to frame the top and bottom part of the cloud, and set by y.range.split and y.step.factor to frame the left and right part of the cloud*

*# If a second data frame is provided, corresponding to the data set of a scatterplot (with a x column for x-axis values and a y column for the y-axis column), then fun\_segmentation() defines the dots of this data frame outside of the frame of the first data frame*

*# ARGUMENTS*

*# data1: a dataframe containing a column of x-axis values and a column of y-axis values*

*# x1: character string of the data1 column name for x-axis (first column of data1 by default)*

*# y1: character string of the data1 column name for y-axis (second column of data1 by default)*

*# x.range.split: positive non null numeric value giving the number of interval on the x value range. if x.range is the range of the dots on the x-axis, then abs(diff(x.range) / x.range.split) gives the window size. Window size decreases when range.split increases. In unit of x-axis. At least one of the x.range.split and y.range.split must be non NULL*

*# x.step.factor: positive non null numeric value giving the shift step of the window. If x.step.factor = 1, no overlap during the sliding (when the window slides from position n to position n+1, no overlap between the two positions). If x.step.factor = 2, 50% of overlap (when the window slides from position n to position n+1, the window on position n+1 overlap 50% of the window when it was on position n)*

*# y.range.split: same as x.range.split for the y-axis. At least one of the x.range.split and y.range.split must be non NULL*

*# y.step.factor: same as x.step.factor for the y-axis*

*# error: proportion (from 0 to 1) of false positives (i.e., proportion of dots from data1 outside of the frame). 0.05 means 5% of the dots from data1 outside of the frame*

*# data2: a dataframe containing a column of x-axis values and a column of y-axis values, for which outside dots of the data1 cloud has to be determined. Write NULL if not required*

*# x2: character string of the data1 column name for x-axis (first column of data1 by default)*

*# y2: character string of the data1 column name for y-axis (second column of data1 by default)*

*# xy.cross.kind: if data2 is non null and if both x.range.split and y.range.split are non null, which dots are finally significants? Write "&" for intersection of outside dots on x and on y. Write "|" for union of outside dots on x and on y*

*# graph.check: logical. Print a pdf graph that check the frame?*

*# graph.path: absolute path of the directory to print the pdf graph.check. Ignored if graph.check is FALSE*

*# path.lib: absolute path of the required packages, if not in the default folders. Ignored if graph.check is FALSE*

*# REQUIRED PACKAGES*

*# ggplot2 if graph.check is TRUE*

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

*# fun\_param\_check()*

*# if graph.check is TRUE:*

*# fun\_pack\_import()*

*# fun\_open\_window()*

*# fun\_gg\_palette()*

*# fun\_gg\_scatter()*

*# fun\_gg\_empty\_graph()*

*# fun\_close\_specif\_window()*

*# RETURN*

*# a pdf plot if graph.check is TRUE*

*# a list containing:*

*# hframe: x and y coordinates of the bottom and top frames for frame plotting (frame1 for the left step and frame2 for the right step)*

*# vframe: x and y coordinates of the left and right frames for frame plotting (frame1 for the down step and frame2 for the top step)*

*# data1.signif.dot: the significant dots of data1 (i.e., dots outside the frame)*

*# data2.signif.dot: the significant dots of data2 if non NULL (i.e., dots outside the frame)*

*# warnings: warning messages*

*# EXAMPLES*

*# set.seed(1) ; data1 = data.frame(x = rnorm(500), y = rnorm(500)) ; data2 = data.frame(x = rnorm(500, 0, 2), y = rnorm(500, 0, 2)) ; set.seed(NULL) ; fun\_segmentation(data1 = data1, x1 = names(data1)[1], y1 = names(data1)[2], x.range.split = 20, x.step.factor = 10, y.range.split = 23, y.step.factor = 10, error = 0, data2 = data2, x2 = names(data2)[1], y2 = names(data2)[2], xy.cross.kind = "|", graph.check = TRUE, graph.path = "C:/Users/Gael/Desktop/", path.lib = NULL)*

*# set.seed(1) ; data1 = data.frame(x = rnorm(500), y = rnorm(500)) ; data2 = data.frame(x = rnorm(500, 0, 2), y = rnorm(500, 0, 2)) ; set.seed(NULL) ; fun\_segmentation(data1 = data1, x1 = names(data1)[1], y1 = names(data1)[2], x.range.split = NULL, x.step.factor = 10, y.range.split = 23, y.step.factor = 10, error = 0, data2 = data2, x2 = names(data2)[1], y2 = names(data2)[2], xy.cross.kind = "|", graph.check = TRUE, graph.path = "C:/Users/Gael/Desktop/", path.lib = NULL)*

*# set.seed(1) ; data1 = data.frame(x = rnorm(500), y = rnorm(500)) ; data2 = data.frame(x = rnorm(500, 0, 2), y = rnorm(500, 0, 2)) ; set.seed(NULL) ; fun\_segmentation(data1 = data1, x1 = names(data1)[1], y1 = names(data1)[2], x.range.split = 20, x.step.factor = 10, y.range.split = NULL, y.step.factor = 10, error = 0, data2 = data2, x2 = names(data2)[1], y2 = names(data2)[2], xy.cross.kind = "&", graph.check = TRUE, graph.path = "C:/Users/Gael/Desktop/", path.lib = NULL)*

*# DEBUGGING*

*# set.seed(1) ; data1 = data.frame(x = rnorm(50), y = rnorm(50)) ; x1 = names(data1)[1] ; y1 = names(data1)[2] ; x.range.split = 5 ; x.step.factor = 10 ; y.range.split = 5 ; y.step.factor = 10 ; error = 0 ; data2 = data.frame(x = rnorm(50, 0, 2), y = rnorm(50, 0, 2)) ; set.seed(NULL) ; x2 = names(data2)[1] ; y2 = names(data2)[2] ; xy.cross.kind = "|" ; graph.check = TRUE ; graph.path = "C:/Users/Gael/Desktop/" ; path.lib = NULL*

*# set.seed(1) ; data1 = data.frame(x = rnorm(500), y = rnorm(500)) ; data2 = data.frame(x = rnorm(500, 0, 2), y = rnorm(500, 0, 2)) ; set.seed(NULL) ; x1 = names(data1)[1] ; y1 = names(data1)[2] ; x.range.split = NULL ; x.step.factor = 10 ; y.range.split = 23 ; y.step.factor = 10 ; error = 0 ; x2 = names(data2)[1] ; y2 = names(data2)[2] ; xy.cross.kind = "|" ; graph.check = TRUE ; graph.path = "C:/Users/Gael/Desktop/" ; path.lib = NULL*

*# set.seed(1) ; data1 = data.frame(x = rnorm(500), y = rnorm(500)) ; data2 = data.frame(x = rnorm(500, 0, 2), y = rnorm(500, 0, 2)) ; set.seed(NULL) ; x1 = names(data1)[1] ; y1 = names(data1)[2] ; x.range.split = 20 ; x.step.factor = 10 ; y.range.split = NULL ; y.step.factor = 10 ; error = 0 ; x2 = names(data2)[1] ; y2 = names(data2)[2] ; xy.cross.kind = "&" ; graph.check = TRUE ; graph.path = "C:/Users/Gael/Desktop/" ; path.lib = NULL*

*# function name*

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

*# end function name*

*# required function checking*

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

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

**stop(tempo.cat)**

**}**

*# end required function checking*

*# argument checking*

**arg.check <- NULL** *# for function debbuging*

**checked.arg.names <- NULL** *# for function debbuging*

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

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

**if(tempo$problem == FALSE & length(data1) < 2){**

**cat(paste0("\n\n============\n\nERROR IN ", function.name, ": data1 ARGUMENT MUST BE A DATA FRAME OF AT LEAST 2 COLUMNS\n\n============\n\n"))**

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

**}**

**tempo <- fun\_param\_check(data = x1, class = "vector", mode = "character", length = 1, fun.name = function.name) ; eval(ee)**

**if(tempo$problem == FALSE & ! (x1 %in% names(data1))){**

**cat(paste0("\n\n================\n\nERROR IN ", function.name, ": x1 ARGUMENT MUST BE A COLUMN NAME OF data1\n\n================\n\n"))**

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

**}**

**tempo <- fun\_param\_check(data = y1, class = "vector", mode = "character", length = 1, fun.name = function.name) ; eval(ee)**

**if(tempo$problem == FALSE & ! (y1 %in% names(data1))){**

**cat(paste0("\n\n================\n\nERROR IN ", function.name, ": y1 ARGUMENT MUST BE A COLUMN NAME OF data1\n\n================\n\n"))**

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

**}**

**if(is.null(x.range.split) & is.null(y.range.split)){**

**cat(paste0("\n\n================\n\nERROR IN ", function.name, ": AT LEAST ONE OF THE x.range.split AND y.range.split ARGUMENTS MUST BE NON NULL\n\n================\n\n"))**

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

**}**

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

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

**if(tempo$problem == FALSE & x.range.split < 1){**

**cat(paste0("\n\n================\n\nERROR IN ", function.name, ": x.range.split ARGUMENT CANNOT BE LOWER THAN 1\n\n================\n\n"))**

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

**}**

**}**

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

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

**if(tempo$problem == FALSE & y.range.split < 1){**

**cat(paste0("\n\n================\n\nERROR IN ", function.name, ": y.range.split ARGUMENT CANNOT BE LOWER THAN 1\n\n================\n\n"))**

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

**}**

**}**

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

**if(tempo$problem == FALSE & x.step.factor < 1){**

**cat(paste0("\n\n================\n\nERROR IN ", function.name, ": x.step.factor ARGUMENT CANNOT BE LOWER THAN 1\n\n================\n\n"))**

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

**}**

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

**if(tempo$problem == FALSE & y.step.factor < 1){**

**cat(paste0("\n\n================\n\nERROR IN ", function.name, ": y.step.factor ARGUMENT CANNOT BE LOWER THAN 1\n\n================\n\n"))**

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

**}**

**tempo <- fun\_param\_check(data = error, prop = TRUE, length = 1, fun.name = function.name) ; eval(ee)**

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

**if(is.null(x2) | is.null(y2)){**

**cat(paste0("\n\n================\n\nERROR IN ", function.name, ": x2 AND y2 ARGUMENTS CANNOT BE NULL IF data2 ARGUMENT IS NON NULL\n\n================\n\n"))**

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

**}**

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

**if(tempo$problem == FALSE & length(data2) < 2){**

**cat(paste0("\n\n============\n\nERROR IN ", function.name, ": data2 ARGUMENT MUST BE A DATA FRAME OF AT LEAST 2 COLUMNS\n\n============\n\n"))**

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

**}**

**if( ! is.null(x2)){**

**tempo <- fun\_param\_check(data = x2, class = "vector", mode = "character", length = 1, fun.name = function.name) ; eval(ee)**

**if(tempo$problem == FALSE & ! (x2 %in% names(data2))){**

**cat(paste0("\n\n================\n\nERROR IN ", function.name, ": x2 ARGUMENT MUST BE A COLUMN NAME OF data2\n\n================\n\n"))**

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

**}**

**}**

**if( ! is.null(y2)){**

**tempo <- fun\_param\_check(data = y2, class = "vector", mode = "character", length = 1, fun.name = function.name) ; eval(ee)**

**if(tempo$problem == FALSE & ! (y2 %in% names(data2))){**

**cat(paste0("\n\n================\n\nERROR IN ", function.name, ": y2 ARGUMENT MUST BE A COLUMN NAME OF data2\n\n================\n\n"))**

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

**}**

**}**

**}**

**if( ! (is.null(x.range.split)) & ! (is.null(y.range.split))){**

**tempo <- fun\_param\_check(data = xy.cross.kind, options = c("&", "|"), length = 1, fun.name = function.name) ; eval(ee)**

**}**

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

**if(tempo$problem == FALSE & graph.check == TRUE){**

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

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

**cat(paste0("\n\n============\n\nERROR IN ", function.name, ": \nDIRECTORY PATH INDICATED IN THE graph.path PARAMETER DOES NOT EXISTS: ", graph.path, "\n\n============\n\n"))**

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

**}**

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

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

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

**cat(paste0("\n\n============\n\nERROR IN ", function.name, ": \nDIRECTORY PATH INDICATED IN THE path.lib PARAMETER DOES NOT EXISTS: ", path.lib, "\n\n============\n\n"))**

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

**}**

**}**

**}**

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

**stop()** *# nothing else because print = TRUE by default in fun\_param\_check()*

**}**

*# 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\_param\_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\_param\_check()*

*# end argument checking*

*# other required function checking*

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

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

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

**stop(tempo.cat)**

**}**

**if(length(find("fun\_open\_window", mode = "function")) == 0){**

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

**stop(tempo.cat)**

**}**

**if(length(find("fun\_gg\_palette", mode = "function")) == 0){**

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

**stop(tempo.cat)**

**}**

**if(length(find("fun\_gg\_empty\_graph", mode = "function")) == 0){**

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

**stop(tempo.cat)**

**}**

**if(length(find("fun\_gg\_scatter", mode = "function")) == 0){**

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

**stop(tempo.cat)**

**}**

**if(length(find("fun\_close\_specif\_window", mode = "function")) == 0){**

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

**stop(tempo.cat)**

**}**

**}**

*# end other required function checking*

*# package checking*

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

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

**}**

*# end package checking*

*# main code*

**warning <- NULL**

**data1 <- data1[ ! duplicated(data1[, c(x1, y1)]), ]** *# remove the dots that have same x and y values*

**data1 <- cbind(data1, DOT\_NB = 1:nrow(data1))**

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

**data2 <- data2[ ! duplicated(data2[, c(x2, y2)]), ]** *# remove the dots that have same x and y values*

**data2 <- cbind(data2, DOT\_NB = 1:nrow(data2))**

**}**

*# may be create vector of each column to increase speed*

*# Method using x unit interval*

**x.data1.l <- NULL** *# x coord of the y upper and lower limits defined on the data1 cloud for left step line*

**x.data1.r <- NULL** *# x coord of the y upper and lower limits defined on the data1 cloud for right step line*

**y.data1.down.limit.l <- NULL** *# lower limit of the data1 cloud for left step line*

**y.data1.top.limit.l <- NULL** *# upper limit of the data1 cloud for left step line*

**y.data1.down.limit.r <- NULL** *# lower limit of the data1 cloud for right step line*

**y.data1.top.limit.r <- NULL** *# upper limit of the data1 cloud for left step line*

**x.range <- range(data1[, x1], na.rm = TRUE)**

**y.range <- range(data1[, y1], na.rm = TRUE)**

**x.range.plot <- range(data1[, x1], na.rm = TRUE)**

**y.range.plot <- range(data1[, y1], na.rm = TRUE)**

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

**x.range.plot <- range(data1[, x1], data2[, x2], na.rm = TRUE)**

**y.range.plot <- range(data1[, y1], data2[, y2], na.rm = TRUE)**

**}**

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

*# data.frame ordering to slide the window from small to big values + sliding window definition*

**data1 <- data1[order(data1[, x1], na.last = TRUE), ]**

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

**data2 <- data2[order(data2[, x2], na.last = TRUE), ]**

**}**

**x.win.size <- abs(diff(x.range) / x.range.split)** *# in unit of x-axis*

**step <- x.win.size / x.step.factor**

*# end data.frame ordering to slide the window from small to big values + sliding window definition*

*# x-axis sliding and y-axis limits of the data1 cloud -> y significant data2*

**loop.nb <- ceiling((diff(x.range) - x.win.size) / step)** *# x.win.size + n \* step covers the x range if x.win.size + n \* step >= diff(x.range), thus if n >= (diff(x.range) - x.win.size) / step*

**y.outside.data1.dot.nb <- integer()** *# vector that will contain the selected rows numbers of data1 that are upper or lower than the frame*

**y.inside.data1.dot.nb <- integer()** *# vector that will contain the selected rows numbers of data1 that are not upper or lower than the frame*

**y.data1.median <- median(data1[, y1], na.rm = TRUE)** *# will be used for sliding windows without data1 in it*

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

**y.outside.data2.dot.nb <- integer()** *# vector that will contain the selected 1D coordinates (i.e., dots) of data2 that are upper or lower than the data1 frame*

**y.inside.data2.dot.nb <- integer()** *# vector that will contain the 1D coordinates (i.e., dots) of data2 that are not upper or lower than the data1 frame*

*# recover data2 dots outside the range of data1*

**if(any(data2[, x2] < x.range[1])){**

**y.outside.data2.dot.nb <- c(y.outside.data2.dot.nb, data2$DOT\_NB[data2[, x2] < x.range[1]])**

**}**

**if(any(data2[, x2] > x.range[2])){**

**y.outside.data2.dot.nb <- c(y.outside.data2.dot.nb, data2$DOT\_NB[data2[, x2] > x.range[2]])**

**}**

*# end recover data2 dots outside the range of data1*

**}**

*# loop.ini.time <- as.numeric(Sys.time())*

**for(i1 in 0:(loop.nb + 1)){**

**min.pos <- x.range[1] + step \* i1** *# lower position of the sliding window in data1*

**max.pos <- min.pos + x.win.size** *# upper position of the sliding window in data1*

**x.data1.l <- c(x.data1.l, min.pos, min.pos + step)** *# min.pos + step to make the steps*

**x.data1.r <- c(x.data1.r, max.pos, max.pos + step)** *# max.pos + step to make the steps*

**x.data1.dot.here <- data1[, x1] >= min.pos & data1[, x1] < max.pos** *# is there data1 dot present in the sliding window, considering the x axis?*

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

**x.data2.dot.here <- data2[, x2] >= min.pos & data2[, x2] < max.pos** *# is there data2 dot present in the sliding window, considering the x axis?*

**}**

*# recover the data1 dots outside the frame*

**if(any(x.data1.dot.here == TRUE)){**

**tempo.y.data1.top.limit <- quantile(data1[x.data1.dot.here, y1], probs = 1 - error, na.rm = TRUE)**

**tempo.y.data1.down.limit <- quantile(data1[x.data1.dot.here, y1], probs = 0 + error, na.rm = TRUE)**

**y.data1.top.limit.l <- c(y.data1.top.limit.l, tempo.y.data1.top.limit, tempo.y.data1.top.limit)**

**y.data1.down.limit.l <- c(y.data1.down.limit.l, tempo.y.data1.down.limit, tempo.y.data1.down.limit)**

**y.data1.top.limit.r <- c(y.data1.top.limit.r, tempo.y.data1.top.limit, tempo.y.data1.top.limit)**

**y.data1.down.limit.r <- c(y.data1.down.limit.r, tempo.y.data1.down.limit, tempo.y.data1.down.limit)**

**y.data1.dot.signif <- ! (data1[, y1] <= tempo.y.data1.top.limit & data1[, y1] >= tempo.y.data1.down.limit) & x.data1.dot.here** *# is there data1 dot present in the sliding window, above or below the data1 limits, considering the y axis?*

**y.data1.dot.not.signif <- x.data1.dot.here & ! y.data1.dot.signif**

**y.outside.data1.dot.nb <- c(y.outside.data1.dot.nb, data1$DOT\_NB[y.data1.dot.signif])** *# recover the row number of data1*

**y.outside.data1.dot.nb <- unique(y.outside.data1.dot.nb)**

**y.inside.data1.dot.nb <- c(y.inside.data1.dot.nb, data1$DOT\_NB[y.data1.dot.not.signif])**

**y.inside.data1.dot.nb <- unique(y.inside.data1.dot.nb)**

**}else{**

**y.data1.top.limit.l <- c(y.data1.top.limit.l, y.data1.median, y.data1.median)**

**y.data1.down.limit.l <- c(y.data1.down.limit.l, y.data1.median, y.data1.median)**

**y.data1.top.limit.r <- c(y.data1.top.limit.r, y.data1.median, y.data1.median)**

**y.data1.down.limit.r <- c(y.data1.down.limit.r, y.data1.median, y.data1.median)**

**}**

*# end recover the data1 dots outside the frame*

*# recover the data2 dots outside the frame*

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

**if(any(x.data1.dot.here == TRUE) & any(x.data2.dot.here == TRUE)){**

**y.data2.dot.signif <- ! (data2[, y2] <= tempo.y.data1.top.limit & data2[, y2] >= tempo.y.data1.down.limit) & x.data2.dot.here** *# is there data2 dot present in the sliding window, above or below the data1 limits, considering the y axis?*

**y.data2.dot.not.signif <- x.data2.dot.here & ! y.data2.dot.signif**

**y.outside.data2.dot.nb <- c(y.outside.data2.dot.nb, data2$DOT\_NB[y.data2.dot.signif])**

**y.outside.data2.dot.nb <- unique(y.outside.data2.dot.nb)**

**y.inside.data2.dot.nb <- c(y.inside.data2.dot.nb, data2$DOT\_NB[y.data2.dot.not.signif])**

**y.inside.data2.dot.nb <- unique(y.inside.data2.dot.nb)**

**}else if(any(x.data1.dot.here == FALSE) & any(x.data2.dot.here == TRUE)){** *# problem: data2 dots in the the windows but no data1 dots to generates the quantiles*

**tempo.warning <- paste0("THE [", round(min.pos, 3), " ; ", round(max.pos, 3), "] INTERVAL DOES NOT CONTAIN data1 X VALUES BUT CONTAINS data2 X VALUES WHICH CANNOT BE EVALUATED.\nTHE CONCERNED data2 NROWS ARE:\n", paste(which(x.data1.dot.here == FALSE & x.data2.dot.here == TRUE), collapse = "\n"))**

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

**}**

**}**

*# end recover the data2 dots outside the frame*

*# if(any(i1 == seq(1, loop.nb, 500))){*

*# loop.fin.time <- as.numeric(Sys.time()) # time of process end*

*# cat(paste0("COMPUTATION TIME OF LOOP ", i1, " / ", loop.nb, ": ", as.character(lubridate::seconds\_to\_period(round(loop.fin.time - loop.ini.time))), "\n"))*

*# }*

**}**

**if(max.pos < x.range[2]){**

**tempo.cat <- paste0("\n\n================\n\nERROR IN ", function.name, ": THE SLIDING WINDOW HAS NOT REACHED THE MAX VALUE OF data1 ON THE X-AXIS: ", max.pos, " VERSUS ", x.range[2], "\n\n================\n\n")**

**stop(tempo.cat)**

**}**

**y.outside.data1.dot.nb.final <- y.outside.data1.dot.nb[ ! (y.outside.data1.dot.nb %in% y.inside.data1.dot.nb)]** *# if a row number of y.inside.data2.dot.nb is present in y.outside.data2.dot.nb, it means that during the sliding, a dot has been sometime inside, sometime outside -> removed*

**y.outside.data2.dot.nb.final <- y.outside.data2.dot.nb[ ! (y.outside.data2.dot.nb %in% y.inside.data2.dot.nb)]** *# if a row number of y.inside.data2.dot.nb is present in y.outside.data2.dot.nb, it means that during the sliding, a dot has been sometime inside, sometime outside -> removed*

*# end x-axis sliding and y-axis limits of the data1 cloud -> y significant data2*

**}**

*# end Method using x unit interval*

*# Method using y unit interval*

**y.data1.d <- NULL** *# y coord of the x upper and lower limits defined on the data1 cloud for down step line*

**y.data1.t <- NULL** *# y coord of the x upper and lower limits defined on the data1 cloud for top step line*

**x.data1.left.limit.d <- NULL** *# left limit of the data1 cloud for down step line*

**x.data1.right.limit.d <- NULL** *# right limit of the data1 cloud for down step line*

**x.data1.left.limit.t <- NULL** *# left limit of the data1 cloud for top step line*

**x.data1.right.limit.t <- NULL** *# right limit of the data1 cloud for top step line*

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

*# data.frame ordering to slide the window from small to big values + sliding window definition*

**data1 <- data1[order(data1[, y1], na.last = TRUE), ]**

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

**data2 <- data2[order(data2[, y2], na.last = TRUE), ]**

**}**

**y.win.size <- abs(diff(y.range) / y.range.split)** *# in unit of y-axis*

**step <- y.win.size / y.step.factor**

*# end data.frame ordering to slide the window from small to big values + sliding window definition*

*# y-axis sliding and x-axis limits of the data1 cloud -> x significant data2*

**loop.nb <- ceiling((diff(y.range) - y.win.size) / step)** *# y.win.size + n \* step covers the y range if y.win.size + n \* step >= diff(y.range), thus if n >= (diff(y.range) - y.win.size) / step*

**x.outside.data1.dot.nb <- integer()** *# vector that will contain the selected rows numbers of data1 that are upper or lower than the frame*

**x.inside.data1.dot.nb <- integer()** *# vector that will contain the selected rows numbers of data1 that are not upper or lower than the frame*

**x.data1.median <- median(data1[, x1], na.rm = TRUE)** *# will be used for sliding windows without data1 in it*

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

**x.outside.data2.dot.nb <- integer()** *# vector that will contain the selected 1D coordinates (i.e., dots) of data2 that are upper or lower than the data1 frame*

**x.inside.data2.dot.nb <- integer()** *# vector that will contain the 1D coordinates (i.e., dots) of data2 that are not upper or lower than the data1 frame*

*# recover data2 dots outside the range of data1*

**if(any(data2[, y2] < y.range[1])){**

**x.outside.data2.dot.nb <- c(x.outside.data2.dot.nb, data2$DOT\_NB[data2[, y2] < y.range[1]])**

**}**

**if(any(data2[, y2] > y.range[2])){**

**x.outside.data2.dot.nb <- c(x.outside.data2.dot.nb, data2$DOT\_NB[data2[, y2] > y.range[2]])**

**}**

*# end recover data2 dots outside the range of data1*

**}**

*# loop.ini.time <- as.numeric(Sys.time())*

**for(i1 in 0:(loop.nb + 1)){**

**min.pos <- y.range[1] + step \* i1** *# lower position of the sliding window in data1*

**max.pos <- min.pos + y.win.size** *# upper position of the sliding window in data1*

**y.data1.d <- c(y.data1.d, min.pos, min.pos + step)** *# min.pos + step to make the steps*

**y.data1.t <- c(y.data1.t, max.pos, max.pos + step)** *# max.pos + step to make the steps*

**y.data1.dot.here <- data1[, y1] >= min.pos & data1[, y1] < max.pos** *# is there data1 dot present in the sliding window, considering the y axis?*

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

**y.data2.dot.here <- data2[, y2] >= min.pos & data2[, y2] < max.pos** *# is there data2 dot present in the sliding window, considering the y axis?*

**}**

*# recover the data1 dots outside the frame*

**if(any(y.data1.dot.here == TRUE)){**

**tempo.x.data1.right.limit <- quantile(data1[y.data1.dot.here, x1], probs = 1 - error, na.rm = TRUE)**

**tempo.x.data1.left.limit <- quantile(data1[y.data1.dot.here, x1], probs = 0 + error, na.rm = TRUE)**

**x.data1.right.limit.d <- c(x.data1.right.limit.d, tempo.x.data1.right.limit, tempo.x.data1.right.limit)**

**x.data1.left.limit.d <- c(x.data1.left.limit.d, tempo.x.data1.left.limit, tempo.x.data1.left.limit)**

**x.data1.right.limit.t <- c(x.data1.right.limit.t, tempo.x.data1.right.limit, tempo.x.data1.right.limit)**

**x.data1.left.limit.t <- c(x.data1.left.limit.t, tempo.x.data1.left.limit, tempo.x.data1.left.limit)**

**x.data1.dot.signif <- ! (data1[, x1] <= tempo.x.data1.right.limit & data1[, x1] >= tempo.x.data1.left.limit) & y.data1.dot.here** *# is there data2 dot present in the sliding window, above or below the data1 limits, considering the x axis?*

**x.data1.dot.not.signif <- y.data1.dot.here & ! x.data1.dot.signif**

**x.outside.data1.dot.nb <- c(x.outside.data1.dot.nb, data1$DOT\_NB[x.data1.dot.signif])** *# recover the row number of data1*

**x.outside.data1.dot.nb <- unique(x.outside.data1.dot.nb)**

**x.inside.data1.dot.nb <- c(x.inside.data1.dot.nb, data1$DOT\_NB[x.data1.dot.not.signif])**

**x.inside.data1.dot.nb <- unique(x.inside.data1.dot.nb)**

**}else{**

**x.data1.right.limit.d <- c(x.data1.right.limit.d, x.data1.median, x.data1.median)**

**x.data1.left.limit.d <- c(x.data1.left.limit.d, x.data1.median, x.data1.median)**

**x.data1.right.limit.t <- c(x.data1.right.limit.t, x.data1.median, x.data1.median)**

**x.data1.left.limit.t <- c(x.data1.left.limit.t, x.data1.median, x.data1.median)**

**}**

*# end recover the data1 dots outside the frame*

*# recover the data2 dots outside the frame*

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

**if(any(y.data1.dot.here == TRUE) & any(y.data2.dot.here == TRUE)){**

**x.data2.dot.signif <- ! (data2[, x2] <= tempo.x.data1.right.limit & data2[, x2] >= tempo.x.data1.left.limit) & y.data2.dot.here** *# is there data2 dot present in the sliding window, above or below the data1 limits, considering the x axis?*

**x.data2.dot.not.signif <- y.data2.dot.here & ! x.data2.dot.signif**

**x.outside.data2.dot.nb <- c(x.outside.data2.dot.nb, data2$DOT\_NB[x.data2.dot.signif])**

**x.outside.data2.dot.nb <- unique(x.outside.data2.dot.nb)**

**x.inside.data2.dot.nb <- c(x.inside.data2.dot.nb, data2$DOT\_NB[x.data2.dot.not.signif])**

**x.inside.data2.dot.nb <- unique(x.inside.data2.dot.nb)**

**}else if(any(y.data1.dot.here == FALSE) & any(y.data2.dot.here == TRUE)){** *# recover the data2 dots outside the range of the data1 cloud*

**tempo.warning <- paste0("THE [", round(min.pos, 3), " ; ", round(max.pos, 3), "] INTERVAL DOES NOT CONTAIN data1 Y VALUES BUT CONTAINS data2 Y VALUES WHICH CANNOT BE EVALUATED.\nTHE CONCERNED data2 NROWS ARE:\n", paste(which(y.data1.dot.here == FALSE & y.data2.dot.here == TRUE), collapse = "\n"))**

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

**}**

**}**

*# end recover the data2 dots outside the frame*

*# if(any(i1 == seq(1, loop.nb, 500))){*

*# loop.fin.time <- as.numeric(Sys.time()) # time of process end*

*# cat(paste0("COMPUTATION TIME OF LOOP ", i1, " / ", loop.nb, ": ", as.character(lubridate::seconds\_to\_period(round(loop.fin.time - loop.ini.time))), "\n"))*

*# }*

**}**

**if(max.pos < y.range[2]){**

**tempo.cat <- paste0("\n\n================\n\nERROR IN ", function.name, ": THE SLIDING WINDOW HAS NOT REACHED THE MAX VALUE OF data1 ON THE Y-AXIS: ", max.pos, " VERSUS ", y.range[2], "\n\n================\n\n")**

**stop(tempo.cat)**

**}**

**x.outside.data1.dot.nb.final <- x.outside.data1.dot.nb[ ! (x.outside.data1.dot.nb %in% x.inside.data1.dot.nb)]** *# if a row number of x.inside.data2.dot.nb is present in x.outside.data2.dot.nb, it means that during the sliding, a dot has between sometime inside, sometime outside -> removed*

**x.outside.data2.dot.nb.final <- x.outside.data2.dot.nb[ ! (x.outside.data2.dot.nb %in% x.inside.data2.dot.nb)]** *# if a row number of x.inside.data2.dot.nb is present in x.outside.data2.dot.nb, it means that during the sliding, a dot has between sometime inside, sometime outside -> removed*

*# end y-axis sliding and x-axis limits of the data1 cloud -> x significant data2*

**}**

*# end Method using y unit interval*

**hframe = rbind(**

**data.frame(**

**x = if(is.null(x.data1.l)){NULL}else{x.data1.l},**

**y = if(is.null(x.data1.l)){NULL}else{y.data1.down.limit.l},**

**kind = if(is.null(x.data1.l)){NULL}else{"down.frame1"}**

**),**

**data.frame(**

**x = if(is.null(x.data1.r)){NULL}else{x.data1.r},**

**y = if(is.null(x.data1.r)){NULL}else{y.data1.down.limit.r},**

**kind = if(is.null(x.data1.r)){NULL}else{"down.frame2"}**

**),**

**data.frame(**

**x = if(is.null(x.data1.l)){NULL}else{x.data1.l},**

**y = if(is.null(x.data1.l)){NULL}else{y.data1.top.limit.l},**

**kind = if(is.null(x.data1.l)){NULL}else{"top.frame1"}**

**),**

**data.frame(**

**x = if(is.null(x.data1.r)){NULL}else{x.data1.r},**

**y = if(is.null(x.data1.r)){NULL}else{y.data1.top.limit.r},**

**kind = if(is.null(x.data1.r)){NULL}else{"top.frame2"}**

**)**

**)**

**vframe = rbind(**

**data.frame(**

**x = if(is.null(y.data1.d)){NULL}else{x.data1.left.limit.d},**

**y = if(is.null(y.data1.d)){NULL}else{y.data1.d},**

**kind = if(is.null(y.data1.d)){NULL}else{"left.frame1"}**

**),**

**data.frame(**

**x = if(is.null(y.data1.t)){NULL}else{x.data1.left.limit.t},**

**y = if(is.null(y.data1.t)){NULL}else{y.data1.t},**

**kind = if(is.null(y.data1.t)){NULL}else{"left.frame2"}**

**),**

**data.frame(**

**x = if(is.null(y.data1.d)){NULL}else{x.data1.right.limit.d},**

**y = if(is.null(y.data1.d)){NULL}else{y.data1.d},**

**kind = if(is.null(y.data1.d)){NULL}else{"right.frame1"}**

**),**

**data.frame(**

**x = if(is.null(y.data1.t)){NULL}else{x.data1.right.limit.t},**

**y = if(is.null(y.data1.t)){NULL}else{y.data1.t},**

**kind = if(is.null(y.data1.t)){NULL}else{"right.frame2"}**

**)**

**)**

**data1.signif.dot <- NULL**

**data2.signif.dot <- NULL**

**if(( ! is.null(x.range.split)) & ( ! is.null(y.range.split))){**

**if(xy.cross.kind == "|"){**

**if(length(unique(c(x.outside.data1.dot.nb.final, y.outside.data1.dot.nb.final))) > 0){**

**data1.signif.dot <- data1[data1$DOT\_NB %in% unique(c(x.outside.data1.dot.nb.final, y.outside.data1.dot.nb.final)), ]**

**}**

**}else if(xy.cross.kind == "&"){**

**if(sum(x.outside.data1.dot.nb.final %in% y.outside.data1.dot.nb.final) > 0){**

**data1.signif.dot <- data1[data1$DOT\_NB %in% x.outside.data1.dot.nb.final[x.outside.data1.dot.nb.final %in% y.outside.data1.dot.nb.final], ]**

**}**

**}else{**

**tempo.cat <- (paste0("\n\n============\n\nERROR: CODE INCONSISTENCY\n\n============\n\n"))**

**stop(tempo.cat)**

**}**

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

**if(xy.cross.kind == "|"){**

**if(length(unique(c(x.outside.data2.dot.nb.final, y.outside.data2.dot.nb.final))) > 0){**

**data2.signif.dot <- data2[data2$DOT\_NB %in% unique(c(x.outside.data2.dot.nb.final, y.outside.data2.dot.nb.final)), ]**

**}**

**}else if(xy.cross.kind == "&"){**

**if(sum(x.outside.data2.dot.nb.final %in% y.outside.data2.dot.nb.final) > 0){**

**data2.signif.dot <- data2[data2$DOT\_NB %in% x.outside.data2.dot.nb.final[x.outside.data2.dot.nb.final %in% y.outside.data2.dot.nb.final], ]**

**}**

**}else{**

**tempo.cat <- (paste0("\n\n============\n\nERROR: CODE INCONSISTENCY\n\n============\n\n"))**

**stop(tempo.cat)**

**}**

**}**

**}else if(( ! is.null(x.range.split)) & is.null(y.range.split)){**

**if(length(y.outside.data1.dot.nb.final) > 0){**

**data1.signif.dot <- data1[data1$DOT\_NB %in% y.outside.data1.dot.nb.final, ]**

**}**

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

**if(length(y.outside.data2.dot.nb.final) > 0){**

**data2.signif.dot <- data2[data2$DOT\_NB %in% y.outside.data2.dot.nb.final, ]**

**}**

**}**

**}else if(is.null(x.range.split) & ( ! is.null(y.range.split))){**

**if(length(x.outside.data1.dot.nb.final) > 0){**

**data1.signif.dot <- data1[data1$DOT\_NB %in% x.outside.data1.dot.nb.final, ]**

**}**

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

**if(length(x.outside.data2.dot.nb.final) > 0){**

**data2.signif.dot <- data2[data2$DOT\_NB %in% x.outside.data2.dot.nb.final, ]**

**}**

**}**

**}**

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

**fun\_open\_window(pdf.disp = TRUE, path.fun = graph.path, pdf.name.file = "segmentation\_graph")**

**if(( ! is.null(x.range.split)) & ( ! is.null(y.range.split))){**

**tempo.graph <- fun\_gg\_scatter(data1 = list(data1, hframe, vframe), x = list(x1, "x", "x"), y = list(y1, "y", "y"), categ = list(NULL, "kind", "kind"), legend.name = list("data1", "hframe" , "vframe"), color = list(fun\_gg\_palette(2)[2], rep(hsv(h = c(0.1, 0.15), v = c(0.75, 1)), 2), rep(hsv(h = c(0.5, 0.6), v = c(0.9, 1)), 2)), geom = list("geom\_point", "geom\_path", "geom\_path"), title = "DATA1", xlim = x.range.plot, ylim = y.range.plot)**

**if( ! is.null(tempo.graph$warnings)){**

**warning <- paste0(ifelse(is.null(warning), tempo.graph$warnings, paste0(warning, "\n", tempo.graph$warnings)))**

**}**

**if( ! is.null(data1.signif.dot)){**

**tempo.graph <- fun\_gg\_scatter(data1 = list(data1, hframe, vframe, data1.signif.dot), x = list(x1, "x", "x", x1), y = list(y1, "y", "y", y1), categ = list(NULL, "kind", "kind", NULL), legend.name = list("data1", "hframe" , "vframe", "data1.signif.dots"), color = list(fun\_gg\_palette(2)[2], rep(hsv(h = c(0.1, 0.15), v = c(0.75, 1)), 2), rep(hsv(h = c(0.5, 0.6), v = c(0.9, 1)), 2), "black"), geom = list("geom\_point", "geom\_path", "geom\_path", "geom\_point"), title = "DATA1 + DATA1 SIGNIFICANT DOTS", xlim = x.range.plot, ylim = y.range.plot)**

**if( ! is.null(tempo.graph$warnings)){**

**warning <- paste0(ifelse(is.null(warning), tempo.graph$warnings, paste0(warning, "\n", tempo.graph$warnings)))**

**}**

**}else{**

**fun\_gg\_empty\_graph(text = "NO PLOT BECAUSE NO DATA1 DOTS OUTSIDE THE LIMITS", text.size = 12, title = "DATA1 + DATA1 SIGNIFICANT DOTS")**

**}**

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

**tempo.graph <- fun\_gg\_scatter(data1 = list(data1, data2, hframe , vframe), x = list(x1, x2, "x", "x"), y = list(y1, y2, "y", "y"), categ = list(NULL, NULL, "kind", "kind"), legend.name = list("data1", "data2", "hframe" , "vframe"), color = list(fun\_gg\_palette(2)[2], fun\_gg\_palette(2)[1], rep(hsv(h = c(0.1, 0.15), v = c(0.75, 1)), 2), rep(hsv(h = c(0.5, 0.6), v = c(0.9, 1)), 2)), geom = list("geom\_point", "geom\_point", "geom\_path", "geom\_path"), title = "DATA1 + DATA2", xlim = x.range.plot, ylim = y.range.plot)**

**if( ! is.null(tempo.graph$warnings)){**

**warning <- paste0(ifelse(is.null(warning), tempo.graph$warnings, paste0(warning, "\n", tempo.graph$warnings)))**

**}**

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

**tempo.graph <- fun\_gg\_scatter(data1 = list(data1, data2, data2.signif.dot, hframe , vframe), x = list(x1, x2, x2, "x", "x"), y = list(y1, y2, y2, "y", "y"), categ = list(NULL, NULL, NULL, "kind", "kind"), legend.name = list("data1", "data2", "data2.signif.dots", "hframe" , "vframe"), color = list(fun\_gg\_palette(2)[2], fun\_gg\_palette(2)[1], "black", rep(hsv(h = c(0.1, 0.15), v = c(0.75, 1)), 2), rep(hsv(h = c(0.5, 0.6), v = c(0.9, 1)), 2)), geom = list("geom\_point", "geom\_point", "geom\_point", "geom\_path", "geom\_path"), title = "DATA1 + DATA2 + DATA2 SIGNIFICANT DOTS", xlim = x.range.plot, ylim = y.range.plot)**

**if( ! is.null(tempo.graph$warnings)){**

**warning <- paste0(ifelse(is.null(warning), tempo.graph$warnings, paste0(warning, "\n", tempo.graph$warnings)))**

**}**

**}else{**

**fun\_gg\_empty\_graph(text = "NO PLOT BECAUSE NO DATA2 DOTS OUTSIDE THE LIMITS", text.size = 12, title = "DATA1 + DATA2 + DATA2 SIGNIFICANT DOTS")**

**}**

**}**

**}else if(( ! is.null(x.range.split)) & is.null(y.range.split)){**

**tempo.graph <- fun\_gg\_scatter(data1 = list(data1, hframe), x = list(x1, "x"), y = list(y1, "y"), categ = list(NULL, "kind"), legend.name = list("data1", "hframe"), color = list(fun\_gg\_palette(2)[2], rep(hsv(h = c(0.1, 0.15), v = c(0.75, 1)), 2)), geom = list("geom\_point", "geom\_path"), title = "DATA1", xlim = x.range.plot, ylim = y.range.plot)**

**if( ! is.null(tempo.graph$warnings)){**

**warning <- paste0(ifelse(is.null(warning), tempo.graph$warnings, paste0(warning, "\n", tempo.graph$warnings)))**

**}**

**if( ! is.null(data1.signif.dot)){**

**tempo.graph <- fun\_gg\_scatter(data1 = list(data1, hframe, data1.signif.dot), x = list(x1, "x", x1), y = list(y1, "y", y1), categ = list(NULL, "kind", NULL), legend.name = list("data1", "hframe", "data1.signif.dots"), color = list(fun\_gg\_palette(2)[2], rep(hsv(h = c(0.1, 0.15), v = c(0.75, 1)), 2), "black"), geom = list("geom\_point", "geom\_path", "geom\_point"), title = "DATA1 + DATA1 SIGNIFICANT DOTS", xlim = x.range.plot, ylim = y.range.plot)**

**if( ! is.null(tempo.graph$warnings)){**

**warning <- paste0(ifelse(is.null(warning), tempo.graph$warnings, paste0(warning, "\n", tempo.graph$warnings)))**

**}**

**}else{**

**fun\_gg\_empty\_graph(text = "NO PLOT BECAUSE NO DATA1 DOTS OUTSIDE THE LIMITS", text.size = 12, title = "DATA1 + DATA1 SIGNIFICANT DOTS")**

**}**

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

**tempo.graph <- fun\_gg\_scatter(data1 = list(data1, data2, hframe), x = list(x1, x2, "x"), y = list(y1, y2, "y"), categ = list(NULL, NULL, "kind"), legend.name = list("data1", "data2", "hframe"), color = list(fun\_gg\_palette(2)[2], fun\_gg\_palette(2)[1], rep(hsv(h = c(0.1, 0.15), v = c(0.75, 1)), 2)), geom = list("geom\_point", "geom\_point", "geom\_path"), title = "DATA1 + DATA2", xlim = x.range.plot, ylim = y.range.plot)**

**if( ! is.null(tempo.graph$warnings)){**

**warning <- paste0(ifelse(is.null(warning), tempo.graph$warnings, paste0(warning, "\n", tempo.graph$warnings)))**

**}**

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

**tempo.graph <- fun\_gg\_scatter(data1 = list(data1, data2, data2.signif.dot, hframe), x = list(x1, x2, x2, "x"), y = list(y1, y2, y2, "y"), categ = list(NULL, NULL, NULL, "kind"), legend.name = list("data1", "data2", "data2.signif.dots", "hframe"), color = list(fun\_gg\_palette(2)[2], fun\_gg\_palette(2)[1], "black", rep(hsv(h = c(0.1, 0.15), v = c(0.75, 1)), 2)), geom = list("geom\_point", "geom\_point", "geom\_point", "geom\_path"), title = "DATA1 + DATA2 + DATA2 SIGNIFICANT DOTS", xlim = x.range.plot, ylim = y.range.plot)**

**if( ! is.null(tempo.graph$warnings)){**

**warning <- paste0(ifelse(is.null(warning), tempo.graph$warnings, paste0(warning, "\n", tempo.graph$warnings)))**

**}**

**}else{**

**fun\_gg\_empty\_graph(text = "NO PLOT BECAUSE NO DATA2 DOTS OUTSIDE THE LIMITS", text.size = 12, title = "DATA1 + DATA2 + DATA2 SIGNIFICANT DOTS")**

**}**

**}**

**}else if(is.null(x.range.split) & ( ! is.null(y.range.split))){**

**tempo.graph <- fun\_gg\_scatter(data1 = list(data1, vframe), x = list(x1, "x"), y = list(y1, "y"), categ = list(NULL, "kind"), legend.name = list("data1", "vframe"), color = list(fun\_gg\_palette(2)[2], rep(hsv(h = c(0.5, 0.6), v = c(0.9, 1)), 2)), geom = list("geom\_point", "geom\_path"), title = "DATA1", xlim = x.range.plot, ylim = y.range.plot)**

**if( ! is.null(tempo.graph$warnings)){**

**warning <- paste0(ifelse(is.null(warning), tempo.graph$warnings, paste0(warning, "\n", tempo.graph$warnings)))**

**}**

**if( ! is.null(data1.signif.dot)){**

**tempo.graph <- fun\_gg\_scatter(data1 = list(data1, vframe, data1.signif.dot), x = list(x1, "x", x1), y = list(y1, "y", y1), categ = list(NULL, "kind", NULL), legend.name = list("data1", "vframe", "data1.signif.dots"), color = list(fun\_gg\_palette(2)[2], rep(hsv(h = c(0.5, 0.6), v = c(0.9, 1)), 2), "black"), geom = list("geom\_point", "geom\_path", "geom\_point"), title = "DATA1 + DATA1 SIGNIFICANT DOTS", xlim = x.range.plot, ylim = y.range.plot)**

**if( ! is.null(tempo.graph$warnings)){**

**warning <- paste0(ifelse(is.null(warning), tempo.graph$warnings, paste0(warning, "\n", tempo.graph$warnings)))**

**}**

**}else{**

**fun\_gg\_empty\_graph(text = "NO PLOT BECAUSE NO DATA1 DOTS OUTSIDE THE LIMITS", text.size = 12, title = "DATA1 + DATA1 SIGNIFICANT DOTS")**

**}**

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

**tempo.graph <- fun\_gg\_scatter(data1 = list(data1, data2, vframe), x = list(x1, x2, "x"), y = list(y1, y2, "y"), categ = list(NULL, NULL, "kind"), legend.name = list("data1", "data2", "vframe"), color = list(fun\_gg\_palette(2)[2], fun\_gg\_palette(2)[1], rep(hsv(h = c(0.5, 0.6), v = c(0.9, 1)), 2)), geom = list("geom\_point", "geom\_point", "geom\_path"), title = "DATA1 + DATA2", xlim = x.range.plot, ylim = y.range.plot)**

**if( ! is.null(tempo.graph$warnings)){**

**warning <- paste0(ifelse(is.null(warning), tempo.graph$warnings, paste0(warning, "\n", tempo.graph$warnings)))**

**}**

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

**tempo.graph <- fun\_gg\_scatter(data1 = list(data1, data2, data2.signif.dot, vframe), x = list(x1, x2, x2, "x"), y = list(y1, y2, y2, "y"), categ = list(NULL, NULL, NULL, "kind"), legend.name = list("data1", "data2", "data2.signif.dots", "vframe"), color = list(fun\_gg\_palette(2)[2], fun\_gg\_palette(2)[1], "black", rep(hsv(h = c(0.5, 0.6), v = c(0.9, 1)), 2)), geom = list("geom\_point", "geom\_point", "geom\_point", "geom\_path"), title = "DATA1 + DATA2 + DATA2 SIGNIFICANT DOTS", xlim = x.range.plot, ylim = y.range.plot)**

**if( ! is.null(tempo.graph$warnings)){**

**warning <- paste0(ifelse(is.null(warning), tempo.graph$warnings, paste0(warning, "\n", tempo.graph$warnings)))**

**}**

**}else{**

**fun\_gg\_empty\_graph(text = "NO PLOT BECAUSE NO DATA2 DOTS OUTSIDE THE LIMITS", text.size = 12, title = "DATA1 + DATA2 + DATA2 SIGNIFICANT DOTS")**

**}**

**}**

**}**

**fun\_close\_specif\_window()**

**}**

**tempo.list <- list(hframe = hframe, vframe = vframe, data1.signif.dot = data1.signif.dot, data2.signif.dot = data2.signif.dot, warnings = warning)**

**return(tempo.list)**

**}**

################ Import

######## fun\_pack\_import() #### Check if R packages are present and import into the working environment

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

**fun\_pack\_import <- function(req.package, path.lib = NULL){**

*# AIM*

*# check if the specified R packages are present in the computer and import them into the working environment*

*# ARGUMENTS*

*# req.package: character vector of package names to import*

*# path.lib: optional character vector specifying the absolute pathways of the directories containing some of the listed packages*

*# REQUIRED PACKAGES*

*# none*

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

*# fun\_param\_check()*

*# RETURN*

*# nothing*

*# EXAMPLES*

*# fun\_pack\_import(req.package = "nopackage")*

*# fun\_pack\_import(req.package = "ggplot2")*

*# fun\_pack\_import(req.package = "ggplot2", path.lib = "blablabla")*

*# DEBUGGING*

*# req.package = "ggplot2" ; path.lib = "C:/Program Files/R/R-3.5.1/library"*

*# function name*

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

*# end function name*

*# required function checking*

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

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

**stop(tempo.cat)**

**}**

*# end required function checking*

*# argument checking*

**arg.check <- NULL** *# for function debbuging*

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

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

**tempo <- fun\_param\_check(data = req.package, class = "character", fun.name = function.name) ; eval(ee)**

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

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

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

**cat(paste0("\n\n============\n\nERROR IN ", function.name, ": \nDIRECTORY PATH INDICATED IN THE path.lib PARAMETER DOES NOT EXISTS: ", path.lib, "\n\n============\n\n"))**

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

**}**

**}**

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

**stop()** *# nothing else because print = TRUE by default in fun\_param\_check()*

**}**

*# 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\_param\_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\_param\_check()*

*# end argument checking*

*# main code*

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

**path.lib <- .libPaths()** *# .libPaths(new = path.lib) # or .libPaths(new = c(.libPaths(), path.lib))*

**}else{**

**.libPaths(new = sub(x = path.lib, pattern = "/$|\\\\$", replacement = ""))** *# .libPaths(new = ) add path to default path. Beware: .libPaths() does not support / at the end of a submitted path. Thus check and replace last / or \\ in path*

**}**

**for(i0 in 1:length(req.package)){**

**if( ! req.package[i0] %in% rownames(installed.packages(lib.loc = path.lib))){**

**stop(paste0("\n\n================\n\nERROR IN ", function.name, ": PACKAGE ", req.package[i0], " MUST BE INSTALLED IN:\n", paste(path.lib, collapse = "\n"), "\n\n================\n\n"))**

**}else{**

**suppressWarnings(suppressPackageStartupMessages(library(req.package[i0], lib.loc = path.lib, quietly = TRUE, character.only = TRUE)))**

**}**

**}**

**}**

######## fun\_python\_pack\_import() #### Check if python packages are present

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

**fun\_python\_pack\_import <- function(req.package, path.lib = NULL, R.path.lib = NULL){**

*# AIM*

*# check if the specified python packages are present in the computer (no import)*

*# ARGUMENTS*

*# req.package: character vector of package names to import*

*# path.lib: optional character vector specifying the absolute pathways of the directories containing some of the listed packages*

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

*# REQUIRED PACKAGES*

*# reticulate*

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

*# fun\_param\_check()*

*# fun\_pack\_import()*

*# RETURN*

*# nothing*

*# EXAMPLES*

*# fun\_python\_pack\_import(req.package = "nopackage")*

*# fun\_python\_pack\_import(req.package = "serpentine")*

*# fun\_python\_pack\_import(req.package = "serpentine", path.lib = "blablabla")*

*# DEBUGGING*

*# req.package = "serpentine" ; path.lib = "C:/Program Files/R/R-3.5.1/library" ; R.path.lib = NULL*

*# req.package = "bad" ; path.lib = NULL ; R.path.lib = NULL*

*# function name*

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

*# end function name*

*# required function checking*

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

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

**stop(tempo.cat)**

**}**

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

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

**stop(tempo.cat)**

**}**

*# end required function checking*

*# argument checking*

**arg.check <- NULL** *# for function debbuging*

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

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

**tempo <- fun\_param\_check(data = req.package, class = "character", fun.name = function.name) ; eval(ee)**

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

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

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

**cat(paste0("\n\n============\n\nERROR IN ", function.name, ": \nDIRECTORY PATH INDICATED IN THE path.lib PARAMETER DOES NOT EXISTS: ", path.lib, "\n\n============\n\n"))**

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

**}**

**}**

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

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

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

**cat(paste0("\n\n============\n\nERROR IN ", function.name, ": \nDIRECTORY PATH INDICATED IN THE R.path.lib PARAMETER DOES NOT EXISTS: ", R.path.lib, "\n\n============\n\n"))**

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

**}**

**}**

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

**stop()** *# nothing else because print = TRUE by default in fun\_param\_check()*

**}**

*# 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\_param\_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\_param\_check()*

*# end argument checking*

*# package checking*

**fun\_pack\_import(req.package = "reticulate", path.lib = R.path.lib)**

*# end package checking*

*# main code*

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

**path.lib <- reticulate::py\_run\_string("**

**import sys ;**

**path\_lib = sys.path**

**")** *# python string*

**path.lib <- path.lib$path\_lib**

**}**

**for(i0 in 1:length(req.package)){**

**tempo.try <- vector("list", length = length(path.lib))**

**for(i1 in 1:length(path.lib)){**

**tempo.try[[i1]] <- try(suppressWarnings(reticulate::import\_from\_path(req.package[i0], path = path.lib[i1])), silent = TRUE)**

**}**

**if(all(sapply(tempo.try, FUN = grepl, pattern = "[Ee]rror"))){**

**stop(paste0("\n\n================\n\nERROR IN ", function.name, ": PACKAGE ", req.package[i0], " MUST BE INSTALLED IN THE MENTIONNED DIRECTORY:\n", paste(path.lib, collapse = "\n"), "\n\n================\n\n"))**

**}else{**

**assign(req.package[i0], reticulate::import(req.package[i0]))**

**}**

**}**

**}**

################ Exporting results (text & tables)

######## fun\_export\_data() #### Print string or data object into output file

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

**fun\_export\_data <- function(data = NULL, output ="results.txt", path = "C:/Users/Gael/Desktop", no.overwrite = TRUE, rownames.kept = FALSE, vector.cat = FALSE, noquote = TRUE, sep = 2){**

*# AIM*

*# log file function: print a character string or a data object into a same output file*

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

*# fun\_param\_check()*

*# ARGUMENTS*

*# data: object to print in the output file. cannot be NULL*

*# output: name of the output file*

*# path: location of the output file*

*# no.overwrite: (logical) if output file already exists, defines if the printing is appended (default TRUE) or if the output file content is erased before printing (FALSE)*

*# rownames.kept: (logical) defines whether row names have to be removed or not in small tables (less than length.rows rows)*

*# vector.cat (logical). If TRUE print a vector of length > 1 using cat() instead of capture.output(). Otherwise (default FALSE) the opposite*

*# noquote: (logical). If TRUE no quote are present for the characters*

*# sep: number of separating lines after printed data (must be integer)*

*# RETURN*

*# nothing*

*# EXAMPLES*

*# fun\_export\_data()*

*# fun\_export\_data(data = 1:3, output = "results.txt", path = "C:/Users/Gael/Desktop", no.overwrite = TRUE, rownames.kept = FALSE, vector.cat = FALSE, noquote = FALSE, sep = 2)*

*# DEBUGGING*

*# data = 1:3 ; output = "results.txt" ; path = "C:/Users/Gael/Desktop" ; no.overwrite = TRUE ; rownames.kept = FALSE ; vector.cat = FALSE ; noquote = FALSE ; sep = 2 # for function debugging*

*# function name*

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

*# end function name*

*# required function checking*

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

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

**stop(tempo.cat)**

**}**

*# end required function checking*

*# argument checking*

*# argument checking without fun\_param\_check()*

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

**tempo.cat <- paste0("\n\n================\n\nERROR IN ", function.name, ": data ARGUMENT CANNOT BE NULL\n\n================\n\n")**

**stop(tempo.cat)**

**}**

*# end argument checking without fun\_param\_check()*

*# argument checking with fun\_param\_check()*

**arg.check <- NULL** *# for function debbuging*

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

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

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

**if(tempo$problem == FALSE & output == ""){**

**tempo.cat <- paste0("\n\n================\n\nERROR IN ", function.name, ": output ARGUMENT AS \"\" DOES NOT CORRESPOND TO A VALID FILE NAME\n\n================\n\n")**

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

**}**

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

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

**tempo.cat <- paste0("\n\n================\n\nERROR IN ", function.name, ": path ARGUMENT DOES NOT CORRESPOND TO EXISTING DIRECTORY\n", paste(path, collapse = "\n"),"\n\n================\n\n")**

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

**}**

**tempo <- fun\_param\_check(data = no.overwrite, class = "logical", length = 1, fun.name = function.name) ; eval(ee)**

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

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

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

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

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

**stop()** *# nothing else because print = TRUE by default in fun\_param\_check()*

**}**

*# end argument checking with fun\_param\_check()*

*# 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\_param\_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\_param\_check()*

*# the 4 next lines are inactivated but kept because at a time, I might have a problem with data (solved with data = NULL). These 4 lines are just to know how to detect a missing argument. Important here because if data is not provided, print the code of the data function*

**# arg.user.list <- as.list(match.call(expand.dots=FALSE))[-1]** *# recover all the arguments provided by the function user (excluding the argument with defaults values not provided by the user. Thus, it is really the list indicated by the user)*

**# default.arg.list <- formals(fun = sys.function(sys.parent()))** *# list of all the arguments of the function with their default values (not the values of the user !). It seems that ls() as first line of the function provide the names of the arguments (empty, called, etc., or not)*

**# arg.without.default.value <- sapply(default.arg.list, is.symbol) & sapply(sapply(default.arg.list, as.character), identical, "")** *# logical to detect argument without default values (these are typeof "symbol" and class "name" and empty character*

**# if( ! all(names(default.arg.list)[arg.without.default.value] %in% names(arg.user.list))){** *# test that the arguments with no null values are provided by the user*

**# tempo.cat <- paste0("\n\n================\n\nERROR IN ", function.name, ": VALUE REQUIRED FOR THESE ARGUMENTS WITH NO DEFAULTS VALUES: ", paste(names(default.arg.list)[arg.without.default.value][ ! names(default.arg.list)[arg.without.default.value] %in% names(arg.user.list)], collapse = " "), "\n\n================\n\n")**

**#stop(tempo.cat)**

**# }**

*# end argument checking*

*# main code*

**if(all(class(data) %in% c("matrix", "data.frame", "table"))){**

**if(rownames.kept == FALSE & all(class(data) == "data.frame") & nrow(data) != 0 & nrow(data) <= 4){** *# for data frames with nrows <= 4*

**rownames.output.tables <- ""**

**length.rows <- nrow(data)**

**for(i in 1:length.rows){** *# replace the rownames of the first 4 rows by increasing number of spaces (beacause identical row names not allowed in data frames). This method cannot be extended to more rows as the printed data frame is shifted on the right because of "big empty rownames"*

**rownames.output.tables <- c(rownames.output.tables, paste0(rownames.output.tables[i]," ", collapse=""))**

**}**

**row.names(data) <- rownames.output.tables[1:length.rows]**

**}else if(rownames.kept == FALSE & all(class(data) %in% c("matrix", "table"))){**

**rownames(data) <- rep("", nrow(data))** *# identical row names allowed in matrices and tables*

**}**

**if(noquote == TRUE){**

**capture.output(noquote(data), file=paste0(path, "/", output), append = no.overwrite)**

**}else{**

**capture.output(data, file=paste0(path, "/", output), append = no.overwrite)**

**}**

**}else if(is.vector(data) & all(class(data) != "list") & (length(data) == 1 | vector.cat == TRUE)){**

**if(noquote == TRUE){**

**cat(noquote(data), file= paste0(path, "/", output), append = no.overwrite)**

**}else{**

**cat(data, file= paste0(path, "/", output), append = no.overwrite)**

**}**

**}else{** *# other (array, list, factor or vector with vector.cat = FALSE)*

**if(noquote == TRUE){**

**capture.output(noquote(data), file=paste0(path, "/", output), append = no.overwrite)**

**}else{**

**capture.output(data, file=paste0(path, "/", output), append = no.overwrite)**

**}**

**}**

**sep.final <- paste0(rep("\n", sep), collapse = "")**

**write(sep.final, file= paste0(path, "/", output), append = TRUE)** *# add a sep*

**}**