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

**## ##**

**## CUTE FUNCTIONS ##**

**## ##**

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

**## ##**

**## ##**

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

**# color palette: see https://github.com/EmilHvitfeldt/r-color-palettes**

**# https://usethis.r-lib.org/ and usethat also**

**# change everywhere: if( ! is.null(arg.check)){**

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

**# update graphic examples with good comment, as in barplot**

**# Make a "first round" check for each function if required**

**# Update all argument description, saying, character vector, etc.**

**# check all the functions using fun\_test**

**# check all(, na.rm = TRUE) and any(, na.rm = TRUE)**

**# fun\_mat\_fill does not recognize half matrix anymore**

**# Templates: https://prettydoc.statr.me/themes.html**

**# # package: http://r-pkgs.had.co.nz/**

**# https://pkgdown.r-lib.org/**

**# https://rdrr.io/github/gastonstat/cointoss/**

**# doc:https://www.sphinx-doc.org/en/master/man/sphinx-autogen.html considering that https://www.ericholscher.com/blog/2014/feb/11/sphinx-isnt-just-for-python/**

**# https://docs.readthedocs.io/en/stable/intro/getting-started-with-sphinx.html**

**# https://docs.gitlab.com/ee/user/project/pages/**

**# also register into biotools**

**# For heatmap: see https://bioinfo-fr.net/creer-des-heatmaps-a-partir-de-grosses-matrices-en-r**

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

[################ Object analysis 2](#_Toc50478566)

[######## fun\_check() #### check class, type, length, etc., of objects 2](#_Toc50478567)

[######## fun\_secu() #### verif that local variables are not present in other envs 10](#_Toc50478568)

[######## fun\_info() #### recover object information 12](#_Toc50478569)

[######## fun\_head() #### head of the left or right of big 2D objects 14](#_Toc50478570)

[######## fun\_tail() #### tail of the left or right of big 2D objects 15](#_Toc50478571)

[######## fun\_comp\_1d() #### comparison of two 1D datasets (vectors, factors, 1D tables) 16](#_Toc50478572)

[######## fun\_comp\_2d() #### comparison of two 2D datasets (row & col names, dimensions, etc.) 20](#_Toc50478573)

[######## fun\_comp\_list() #### comparison of two lists 26](#_Toc50478574)

[######## fun\_test() #### test combinations of argument values of a function and return errors (and graphs) 29](#_Toc50478575)

[################ Object modification 43](#_Toc50478576)

[######## fun\_name\_change() #### check a vector of character strings and modify any string if present in another vector 43](#_Toc50478577)

[######## fun\_df\_remod() #### remodeling a data frame to have column name as a qualitative values and vice-versa 45](#_Toc50478578)

[######## fun\_round() #### rounding number if decimal present 47](#_Toc50478579)

[######## fun\_mat\_rotate() #### 90° clockwise matrix rotation 49](#_Toc50478580)

[######## fun\_mat\_num2color() #### convert a numeric matrix into hexadecimal color matrix 50](#_Toc50478581)

[######## fun\_mat\_op() #### assemble several matrices with operation 53](#_Toc50478582)

[######## fun\_mat\_inv() #### return the inverse of a square matrix 56](#_Toc50478583)

[######## fun\_mat\_fill() #### fill the empty half part of a symmetric square matrix 57](#_Toc50478584)

[######## fun\_permut() #### progressively breaks a vector order 61](#_Toc50478585)

[######## fun\_slide() #### return a computation made on a vector using a sliding window 71](#_Toc50478586)

[################ Graphics management 78](#_Toc50478587)

[######## fun\_width() #### window width depending on classes to plot 78](#_Toc50478588)

[######## fun\_open() #### open a GUI or pdf graphic window 80](#_Toc50478589)

[######## fun\_prior\_plot() #### set graph param before plotting (erase axes for instance) 84](#_Toc50478590)

[######## fun\_scale() #### select nice label numbers when setting number of ticks on an axis 88](#_Toc50478591)

[######## fun\_inter\_ticks() #### define coordinates of secondary ticks 93](#_Toc50478592)

[######## fun\_post\_plot() #### set graph param after plotting (axes redesign for instance) 97](#_Toc50478593)

[######## fun\_close() #### close specific graphic windows 108](#_Toc50478594)

[################ Standard graphics 110](#_Toc50478595)

[######## fun\_empty\_graph() #### text to display for empty graphs 110](#_Toc50478596)

[################ gg graphics 112](#_Toc50478597)

[######## fun\_gg\_palette() #### ggplot2 default color palette 112](#_Toc50478598)

[######## fun\_gg\_just() #### ggplot2 justification of the axis labeling, depending on angle 113](#_Toc50478599)

[######## fun\_gg\_get\_legend() #### get the legend of ggplot objects 116](#_Toc50478600)

[######## fun\_gg\_point\_rast() #### ggplot2 raster scatterplot layer 118](#_Toc50478601)

[######## fun\_gg\_boxplot() #### ggplot2 boxplot + background dots if required 121](#_Toc50478602)

[######## fun\_gg\_scatter() #### ggplot2 scatterplot + lines (up to 6 overlays totally) 121](#_Toc50478603)

[######## fun\_gg\_heatmap() #### ggplot2 heatmap + overlaid mask if required 122](#_Toc50478604)

[######## fun\_gg\_empty\_graph() #### text to display for empty graphs 135](#_Toc50478605)

[################ Graphic extraction 137](#_Toc50478606)

[######## fun\_trim() #### display values from a quantitative variable and trim according to defined cut-offs 137](#_Toc50478607)

[######## fun\_segmentation() #### segment a dot cloud on a scatterplot and define the dots from another cloud outside the segmentation 146](#_Toc50478608)

[################ Import 178](#_Toc50478609)

[######## fun\_pack() #### check if R packages are present and import into the working environment 178](#_Toc50478610)

[######## fun\_python\_pack() #### check if python packages are present 180](#_Toc50478611)

[################ Print / Exporting results (text & tables) 182](#_Toc50478612)

[######## fun\_report() #### print string or data object into output file 182](#_Toc50478613)

[######## fun\_get\_message() #### return error/warning/other messages of an expression (that can be exported) 185](#_Toc50478614)

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

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

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

**# Check r\_debugging\_tools-v1.2.R OK**

**# Check fun\_test() (see cute\_checks.docx) Ok**

**# check manual: example to scan again**

**# clear to go Apollo**

**fun\_check <- function(data, data.name = NULL, class = NULL, typeof = NULL, mode = NULL, length = NULL, prop = FALSE, double.as.integer.allowed = FALSE, options = NULL, all.options.in.data = FALSE, na.contain = FALSE, neg.values = TRUE, print = FALSE, 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 or type or mode or length argument must be non null*

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

*# WARNINGS*

*# The function tests what is written in arguments, even if what is written in incoherent. For instance, fun\_check(data = factor(1), class = "factor", mode = "character") will return a problem, and this, what ever the object tested in the data argument, because no object can be class "factor" and mode "character" (factors are class "factor" and mode "numeric")*

*# Since R >= 4.0.0, class(matrix()) returns "matrix" "array", and not "matrix" alone as before. However, use argument class = "matrix" to check for matrix object (of class "matrix" "array" in R >= 4.0.0) and use argument class = "array" to check for array object (of class "array" in R >= 4.0.0)*

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

*# none*

*# ARGUMENTS*

*# data: object to test*

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

*# class: character string. Either one of the class() result (But see the warning section above) or "vector" or "ggplot2" (i.e., objects of class c("gg", "ggplot")) or NULL*

*# typeof: character string. Either one of the typeof() result or NULL*

*# mode: character string. Either one of the mode() result (for non vector object) or NULL*

*# length: numeric value indicating the length of the object. Not considered if NULL*

*# prop: logical. Are the numeric values between 0 and 1 (proportion)? If TRUE, can be used alone, without considering class, etc.*

*# 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 strictly have 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. WARNING: data%%1 == 0 but not isTRUE(all.equal(data%%1, 0)) is used here because the argument checks for integers stored as double (does not check for decimal numbers that are approximate integers)*

*# options: a vector of character strings indicating all the possible option values for data*

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

*# na.contain: logical. Can data contain NA?*

*# neg.values: logical. Are negative numeric values authorized? BEWARE: only considered if set to FALSE, to check for non negative values when class is set to "vector", "numeric", "matrix", "array", "data.frame", "table", or typeof is set to "double", "integer", or mode is set to "numeric". Ignored in other cases, notably with prop argument*

*# print: logical. Print the error message if $problem is TRUE? WARNING: set by default to FALSE, which facilitates the control of the error message output when using fun\_check() inside functions. See the example section*

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

*# RETURN*

*# a list containing:*

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

*# $text: the problem detected*

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

*# EXAMPLES*

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

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

*# argument print with and without assignation*

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

*# test <- 1:3 ; tempo <- fun\_check(data = test, print = FALSE, class = "vector", mode = "character") # the assignation allows to recover a problem without printing it*

*# if(tempo$problem == TRUE){cat(paste0("\n\n================\n\n", tempo$text, "\n\n================\n\n"))}*

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

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

*# DEBUGGING*

*# data = expression(TEST) ; data.name = NULL ; class = "vector" ; typeof = NULL ; mode = NULL ; length = 1 ; prop = FALSE ; double.as.integer.allowed = FALSE ; options = NULL ; all.options.in.data = FALSE ; na.contain = FALSE ; neg.values = TRUE ; print = TRUE ; fun.name = NULL*

*# function name: no used in this function for the error message, to avoid env colliding*

*# argument checking*

*# fun.name checked first because required next*

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

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

**tempo.cat <- paste0("ERROR IN fun\_check(): THE fun.name ARGUMENT MUST BE A CHARACTER VECTOR OF LENGTH 1: ", paste(fun.name, collapse = " "))**

**stop(paste0("\n\n================\n\n", tempo.cat, "\n\n================\n\n"), call. = FALSE)** *# == in stop() to be able to add several messages between ==*

**}**

**}**

*# end fun.name checked first because required next*

*# arg with no default values*

**if(missing(data)){**

**tempo.cat <- paste0("ERROR IN fun\_check()", ifelse(is.null(fun.name), "", paste0(" IN ", fun.name)), ": ARGUMENT data HAS NO DEFAULT VALUE AND REQUIRES ONE")**

**stop(paste0("\n\n================\n\n", tempo.cat, "\n\n================\n\n"), call. = FALSE)** *# == in stop() to be able to add several messages between ==*

**}**

*# end arg with no default values*

*# dealing with NA*

**if(any(is.na(data.name)) | any(is.na(class)) | any(is.na(typeof)) | any(is.na(mode)) | any(is.na(length)) | any(is.na(prop)) | any(is.na(double.as.integer.allowed)) | any(is.na(all.options.in.data)) | any(is.na(na.contain)) | any(is.na(neg.values)) | any(is.na(print)) | any(is.na(fun.name))){**

**tempo.cat <- paste0("ERROR IN fun\_check()", ifelse(is.null(fun.name), "", paste0(" IN ", fun.name)), ": NO ARGUMENT EXCEPT data AND options CAN HAVE NA VALUES\nPROBLEMATIC ARGUMENTS ARE: ", paste(c("data.name", "class", "typeof", "mode", "length", "prop", "double.as.integer.allowed", "all.options.in.data", "na.contain", "neg.values", "print", "fun.name")[c(any(is.na(data.name)), any(is.na(class)), any(is.na(typeof)), any(is.na(mode)), any(is.na(length)), any(is.na(prop)), any(is.na(double.as.integer.allowed)), any(is.na(all.options.in.data)), any(is.na(na.contain)), any(is.na(neg.values)), any(is.na(print)), any(is.na(fun.name)))], collapse = " "))**

**stop(paste0("\n\n================\n\n", tempo.cat, "\n\n================\n\n"), call. = FALSE)** *# == in stop() to be able to add several messages between ==*

**}**

*# end dealing with NA*

*# dealing with NULL*

**if(is.null(prop) | is.null(double.as.integer.allowed) | is.null(all.options.in.data) | is.null(na.contain) | is.null(neg.values) | is.null(print)){**

**tempo.cat <- paste0("ERROR IN fun\_check()", ifelse(is.null(fun.name), "", paste0(" IN ", fun.name)), ": THESE ARGUMENTS\nprop\ndouble.as.integer.allowed\nall.options.in.data\nna.contain\nneg.values\nprint\nCANNOT BE NULL\nPROBLEMATIC ARGUMENTS ARE: ", paste(c("prop", "double.as.integer.allowed", "all.options.in.data", "na.contain", "neg.values", "print")[c(is.null(prop), is.null(double.as.integer.allowed), is.null(all.options.in.data), is.null(na.contain), is.null(neg.values), is.null(print))], collapse = " "))**

**stop(paste0("\n\n================\n\n", tempo.cat, "\n\n================\n\n"), call. = FALSE)** *# == in stop() to be able to add several messages between ==*

**}**

*# end dealing with NULL*

*# dealing with logical*

*# tested below*

*# end dealing with logical*

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

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

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

**stop(paste0("\n\n================\n\n", tempo.cat, "\n\n================\n\n"), call. = FALSE)** *# == in stop() to be able to add several messages between ==*

**}**

**}**

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

**tempo.cat <- paste0("ERROR IN fun\_check()", ifelse(is.null(fun.name), "", paste0(" IN ", fun.name)), ": AT LEAST ONE OF THE options, class, typeof, mode, prop, OR length ARGUMENT MUST BE SPECIFIED (I.E, TRUE FOR prop)")**

**stop(paste0("\n\n================\n\n", tempo.cat, "\n\n================\n\n"), call. = FALSE)** *# == in stop() to be able to add several messages between ==*

**}**

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

**tempo.cat <- paste0("ERROR IN fun\_check()", ifelse(is.null(fun.name), "", paste0(" IN ", fun.name)), ": THE class, typeof, mode ARGUMENTS MUST BE NULL, AND prop FALSE, IF THE options ARGUMENT IS SPECIFIED\nTHE options ARGUMENT MUST BE NULL IF THE class AND/OR typeof AND/OR mode AND/OR prop ARGUMENT IS SPECIFIED")**

**stop(paste0("\n\n================\n\n", tempo.cat, "\n\n================\n\n"), call. = FALSE)** *# == in stop() to be able to add several messages between ==*

**}**

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

**tempo.cat <- paste0("ERROR IN fun\_check()", ifelse(is.null(fun.name), "", paste0(" IN ", fun.name)), ": THE neg.values ARGUMENT MUST BE TRUE OR FALSE ONLY")**

**stop(paste0("\n\n================\n\n", tempo.cat, "\n\n================\n\n"), call. = FALSE)** *# == in stop() to be able to add several messages between ==*

**}**

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

**tempo.cat <- paste0("ERROR IN fun\_check()", ifelse(is.null(fun.name), "", paste0(" IN ", fun.name)), ": THE neg.values ARGUMENT CANNOT BE SWITCHED TO FALSE IF class, typeof AND mode ARGUMENTS ARE NULL")**

**stop(paste0("\n\n================\n\n", tempo.cat, "\n\n================\n\n"), call. = FALSE)** *# == in stop() to be able to add several messages between ==*

**}**

**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", "uneval", "environment", "ggplot2", "ggplot\_built") & any(is.na(class)) != TRUE & length(class) == 1)){** *# length == 1 here because of class(matrix()) since R4.0.0*

**tempo.cat <- paste0("ERROR IN fun\_check()", ifelse(is.null(fun.name), "", paste0(" IN ", fun.name)), ": 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\", \"environment\", \"ggplot2\", \"ggplot\_built\"")**

**stop(paste0("\n\n================\n\n", tempo.cat, "\n\n================\n\n"), call. = FALSE)** *# == in stop() to be able to add several messages between ==*

**}**

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

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

**stop(paste0("\n\n================\n\n", tempo.cat, "\n\n================\n\n"), call. = FALSE)** *# == in stop() to be able to add several messages between ==*

**}**

**}**

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

**if( ! (all(typeof %in% c("logical", "integer", "double", "complex", "character", "list", "expression", "name", "symbol", "closure", "special", "builtin", "environment", "S4")) & length(typeof) == 1 & any(is.na(typeof)) != TRUE)){**

**tempo.cat <- paste0("ERROR IN fun\_check()", ifelse(is.null(fun.name), "", paste0(" IN ", fun.name)), ": typeof ARGUMENT MUST BE ONE OF THESE VALUE:\n\"logical\", \"integer\", \"double\", \"complex\", \"character\", \"list\", \"expression\", \"name\", \"symbol\", \"closure\", \"special\", \"builtin\", \"environment\", \"S4\"")**

**stop(paste0("\n\n================\n\n", tempo.cat, "\n\n================\n\n"), call. = FALSE)** *# == in stop() to be able to add several messages between ==*

**}**

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

**tempo.cat <- paste0("ERROR IN fun\_check()", ifelse(is.null(fun.name), "", paste0(" IN ", fun.name)), ": typeof ARGUMENT CANNOT BE OTHER THAN \"double\" OR \"integer\" IF neg.values ARGUMENT IS SWITCHED TO FALSE")**

**stop(paste0("\n\n================\n\n", tempo.cat, "\n\n================\n\n"), call. = FALSE)** *# == in stop() to be able to add several messages between ==*

**}**

**}**

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

**if( ! (all(mode %in% c("logical", "numeric", "complex", "character", "list", "expression", "name", "symbol", "function", "environment", "S4")) & length(mode) == 1 & any(is.na(mode)) != TRUE)){**

**tempo.cat <- paste0("ERROR IN fun\_check()", ifelse(is.null(fun.name), "", paste0(" IN ", fun.name)), ": mode ARGUMENT MUST BE ONE OF THESE VALUE:\n\"logical\", \"numeric\", \"complex\", \"character\", \"list\", \"expression\", \"name\", \"symbol\", \"function\", \"environment\", \"S4\"")**

**stop(paste0("\n\n================\n\n", tempo.cat, "\n\n================\n\n"), call. = FALSE)** *# == in stop() to be able to add several messages between ==*

**}**

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

**tempo.cat <- paste0("ERROR IN fun\_check()", ifelse(is.null(fun.name), "", paste0(" IN ", fun.name)), ": mode ARGUMENT CANNOT BE OTHER THAN \"numeric\" IF neg.values ARGUMENT IS SWITCHED TO FALSE")**

**stop(paste0("\n\n================\n\n", tempo.cat, "\n\n================\n\n"), call. = FALSE)** *# == in stop() to be able to add several messages between ==*

**}**

**}**

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

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

**tempo.cat <- paste0("ERROR IN fun\_check()", ifelse(is.null(fun.name), "", paste0(" IN ", fun.name)), ": length ARGUMENT MUST BE A SINGLE INTEGER VALUE")**

**stop(paste0("\n\n================\n\n", tempo.cat, "\n\n================\n\n"), call. = FALSE)** *# == in stop() to be able to add several messages between ==*

**}**

**}**

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

**tempo.cat <- paste0("ERROR IN fun\_check()", ifelse(is.null(fun.name), "", paste0(" IN ", fun.name)), ": prop ARGUMENT MUST BE TRUE OR FALSE ONLY")**

**stop(paste0("\n\n================\n\n", tempo.cat, "\n\n================\n\n"), call. = FALSE)** *# == in stop() to be able to add several messages between ==*

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

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

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

**tempo.cat <- paste0("ERROR IN fun\_check()", ifelse(is.null(fun.name), "", paste0(" IN ", fun.name)), ": class ARGUMENT CANNOT BE OTHER THAN NULL, \"vector\", \"numeric\", \"matrix\", \"array\", \"data.frame\", \"table\" IF prop ARGUMENT IS TRUE")** *# not integer because prop*

**stop(paste0("\n\n================\n\n", tempo.cat, "\n\n================\n\n"), call. = FALSE)** *# == in stop() to be able to add several messages between ==*

**}**

**}**

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

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

**tempo.cat <- paste0("ERROR IN fun\_check()", ifelse(is.null(fun.name), "", paste0(" IN ", fun.name)), ": mode ARGUMENT CANNOT BE OTHER THAN NULL OR \"numeric\" IF prop ARGUMENT IS TRUE")**

**stop(paste0("\n\n================\n\n", tempo.cat, "\n\n================\n\n"), call. = FALSE)** *# == in stop() to be able to add several messages between ==*

**}**

**}**

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

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

**tempo.cat <- paste0("ERROR IN fun\_check()", ifelse(is.null(fun.name), "", paste0(" IN ", fun.name)), ": typeof ARGUMENT CANNOT BE OTHER THAN NULL OR \"double\" IF prop ARGUMENT IS TRUE")**

**stop(paste0("\n\n================\n\n", tempo.cat, "\n\n================\n\n"), call. = FALSE)** *# == in stop() to be able to add several messages between ==*

**}**

**}**

**}**

**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("ERROR IN fun\_check()", ifelse(is.null(fun.name), "", paste0(" IN ", fun.name)), ": THE double.as.integer.allowed ARGUMENT MUST BE TRUE OR FALSE ONLY: ", paste(double.as.integer.allowed, collapse = " "))**

**stop(paste0("\n\n================\n\n", tempo.cat, "\n\n================\n\n"), call. = FALSE)** *# == in stop() to be able to add several messages between ==*

**}**

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

**tempo.cat <- paste0("ERROR IN fun\_check()", ifelse(is.null(fun.name), "", paste0(" IN ", fun.name)), ": all.options.in.data ARGUMENT MUST BE A SINGLE LOGICAL VALUE (TRUE OR FALSE ONLY): ", paste(all.options.in.data, collapse = " "))**

**stop(paste0("\n\n================\n\n", tempo.cat, "\n\n================\n\n"), call. = FALSE)** *# == in stop() to be able to add several messages between ==*

**}**

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

**tempo.cat <- paste0("ERROR IN fun\_check(): THE na.contain ARGUMENT MUST BE TRUE OR FALSE ONLY: ", paste(na.contain, collapse = " "))**

**stop(paste0("\n\n================\n\n", tempo.cat, "\n\n================\n\n"), call. = FALSE)** *# == in stop() to be able to add several messages between ==*

**}**

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

**tempo.cat <- paste0("ERROR IN fun\_check(): THE print ARGUMENT MUST BE TRUE OR FALSE ONLY: ", paste(print, collapse = " "))**

**stop(paste0("\n\n================\n\n", tempo.cat, "\n\n================\n\n"), call. = FALSE)** *# == in stop() to be able to add several messages between ==*

**}**

*# fun.name tested at the beginning*

*# 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, na.rm = TRUE)){**

**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 ARE: ", paste(unique(data[ ! (data %in% options)]), collapse = " "))**

**}**

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

**if( ! all(options %in% data)){** *# no need of na.rm = TRUE for all because %in% does not output NA*

**problem <- TRUE**

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

**}**

**}**

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

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

**problem <- TRUE**

**text <- paste0(ifelse(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(class)){**

**if(class == "matrix"){** *# because of class(matric()) since R4.0.0*

**class <- c("matrix", "array")**

**}else if(class == "factor" & all(class(data) %in% c("factor", "ordered"))){** *# to deal with ordered factors*

**class <- c("factor", "ordered")**

**}**

**}**

**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]), " ", if(all(get(arg.names[i2]) %in% c("matrix", "array"))){"matrix"}else if(all(get(arg.names[i2]) %in% c("factor", "ordered"))){"factor"}else{get(arg.names[i2])})**

**'**

*# end script to execute*

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

**if( ! all(data %% 1 == 0, na.rm = TRUE)){** *# to check integers (use %%, meaning the remaining of a division): see the precedent line. isTRUE(all.equal(data%%1, rep(0, length(data)))) not used because we strictly need zero as a result*

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

**}**

**}else if( ! any(all(get(arg.names[i2]) %in% c("vector", "ggplot2"))) & ! all(eval(parse(text = paste0(arg.names[i2], "(data)"))) %in% get(arg.names[i2]))){** *# test the four c("class", "typeof", "mode", "length") arguments with their corresponding function. No need of na.rm = TRUE for all because %in% does not output NA*

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

**}else if(arg.names[i2] == "class" & all(get(arg.names[i2]) == "vector") & ! (all(class(data) %in% "numeric") | all(class(data) %in% "integer") | all(class(data) %in% "character") | all(class(data) %in% "logical"))){** *# test class == "vector". No need of na.rm = TRUE for all because %in% does not output NA*

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

**}else if(arg.names[i2] == "class" & all(get(arg.names[i2]) == "ggplot2") & ! all(class(data) %in% c("gg", "ggplot"))){** *# test ggplot object*

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

**}**

**}**

**}**

**}**

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

**if(is.null(data) | 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(all(class(data) %in% "expression")){ # no need of na.rm = TRUE for all because %in% does not output NA**

**data <- as.character(data)** *# to evaluate the presence of NA*

**}**

**if(na.contain == FALSE & (mode(data) %in% c("logical", "numeric", "complex", "character", "list", "expression", "name", "symbol"))){** *# before it was ! (class(data) %in% c("function", "environment"))*

**if(any(is.na(data)) == TRUE){** *# not on the same line because when data is class envir or function , do not like that*

**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")**

**}**

**}**

**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, fun.name = data.name)**

**return(output)**

**}**

######## fun\_secu() #### verif that local variables are not present in other envs

**fun\_secu <- function(pos = 1, name = NULL){**

*# AIM*

*# verif that local variables are not present in other environments, in order to avoid scope preference usage. The fun\_secu() function checks by default the parent environment. This means that when used inside a function, it checks the local environment of this function. When used in the Global environment, it would check this environment*

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

*# fun\_check()*

*# ARGUMENTS*

*# pos: single integer indicating the position of the environment checked (argument n of parent.frame()). VaLue 1 means one step above the fun\_secu() local environment. Thus, if fun\_secu() is used in the working environment, with pos ==1, variables of this env will be checked in the above envs. If fun\_secu() is used in a function, with pos ==1, variables presents in the local env of the functions will be checked in the above envs (which includes the working environment (Global env)*

*# name: single character string indicating the name of the function checked*

*# RETURN*

*# a character string of the local variables that match variables in the different environments of the R scope, or NULL if no match*

*# EXAMPLES*

*# fun\_secu()*

*# fun\_secu(pos = 2)*

*# mean <- 0 ; fun1 <- function(){sd <- 1 ; fun\_secu(name = as.character(sys.calls()[[length(sys.calls())]]))} ; fun2 <- function(){cor <- 2 ; fun1()} ; fun1() ; fun2() ; rm(mean) # sys.calls() gives the the function name at top stack of the imbricated functions, sys.calls()[[length(sys.calls())]] the name of the just above function. This can also been used for the above function: as.character(sys.call(1))*

*# test.pos <- 2 ; mean <- 0 ; fun1 <- function(){sd <- 1 ; fun\_secu(pos = test.pos, name = if(length(sys.calls()) >= test.pos){as.character(sys.calls()[[length(sys.calls()) + 1 - test.pos]])}else{search()[ (1:length(search()))[test.pos - length(sys.calls())]]})} ; fun2 <- function(){cor <- 2 ; fun1()} ; fun1() ; fun2() ; rm(mean) # for argument name, here is a way to have the name of the tested environment according to test.pos value*

*# DEBUGGING*

*# pos = 1 ; name = 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(utils::find("fun\_check", mode = "function")) == 0){**

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

**stop(paste0("\n\n================\n\n", tempo.cat, "\n\n================\n\n"), call. = FALSE)** *# == in stop() to be able to add several messages between ==*

**}**

*# end required function checking*

*# argument checking*

**arg.check <- NULL #**

**text.check <- NULL #**

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

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

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

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

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

**}**

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

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

**}**

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

*# end argument checking*

*# main code*

*# match.list <- vector("list", length = (length(sys.calls()) - 1 + length(search()) + ifelse(length(sys.calls()) == 1, -1, 0))) # match.list is a list of all the environment tested (local of functions and R envs), length(sys.calls()) - 1 to remove the level of the fun\_secu() function, sys.calls() giving all the names of the imbricated functions, including fun\_secu, ifelse(length(sys.calls()) == 1, -1, 0) to remove Global env if this one is tested*

**tempo.name <- rev(as.character(unlist(sys.calls())))** *# get names of frames (i.e., enclosed env)*

**tempo.frame <- rev(sys.frames())**  *# get frames (i.e., enclosed env)*

*# dealing with source()*

*# source() used in the Global env creates three frames above the Global env, which should be removed because not very interesting for variable duplications. Add a <<-(sys.frames()) in this code and source anova\_contrasts code to see this. With ls(a[[4]]), we can see the content of each env, which are probably elements of source()*

**if(any(sapply(tempo.frame, FUN = environmentName) %in% "R\_GlobalEnv")){**

**global.pos <- which(sapply(tempo.frame, FUN = environmentName) %in% "R\_GlobalEnv")**

*# remove the global env (because already in search(), and all the oabove env*

**tempo.name <- tempo.name[-c(global.pos:length(tempo.frame))]**

**tempo.frame <- tempo.frame[-c(global.pos:length(tempo.frame))]**

**}**

*# end dealing with source()*

**# might have a problem if(length(tempo.name) == 0){**

**match.list <- vector("list", length = length(tempo.name) + length(search()))** *# match.list is a list of all the environment tested (local of functions and R envs), length(sys.calls()) - 1 to remove the level of the fun\_secu() function, sys.calls() giving all the names of the imbricated functions, including fun\_secu, ifelse(length(sys.calls()) == 1, -1, 0) to remove Global env if this one is tested*

**ls.names <- c(tempo.name, search()) # names of the functions + names of the search() environments**

**ls.input <- c(tempo.frame, as.list(search()))** *# environements of the functions + names of the search() environments*

**names(match.list) <- ls.names** *#*

**match.list <- match.list[-c(1:(pos + 1))] # because we check only above pos**

**ls.tested <- ls.input[[pos + 1]]**

**ls.input <- ls.input[-c(1:(pos + 1))]**

**for(i1 in 1:length(match.list)){**

**if(any(ls(name = ls.input[[i1]], all.names = TRUE) %in% ls(name = ls.tested, all.names = TRUE))){**

**match.list[i1] <- list(ls(name = ls.input[[i1]], all.names = TRUE)[ls(name = ls.input[[i1]], all.names = TRUE) %in% ls(name = ls.tested, all.names = TRUE)])**

**}**

**}**

**if( ! all(sapply(match.list, FUN = is.null))){**

**output <- paste0("SOME VARIABLES ", ifelse(is.null(name), "OF THE CHECKED ENVIRONMENT", paste0("OF ", name)), " ARE ALSO PRESENT IN :\n", paste0(names(match.list[ ! sapply(match.list, FUN = is.null)]), ": ", sapply(match.list[ ! sapply(match.list, FUN = is.null)], FUN = paste0, collapse = " "), collapse = "\n"))**

**}else{**

**output <- NULL**

**}**

**return(output)**

**}**

######## fun\_info() #### recover object information

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

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

*# AIM*

*# provide a full description of an object*

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

*# none*

*# ARGUMENTS*

*# data: object to test*

*# RETURN*

*# a list containing information, depending on the class and type of data*

*# if data is made of numerics, provide range, sum, mean, number of NA and number of Inf*

*# please, use names(fun\_info()) and remove what can be too big for easy analysis*

*# EXAMPLES*

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

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

*# fun\_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*

*# function name: no need because no check and no message*

*# 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("NAME" = data.name)**

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

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

**tempo <- list("TYPE" = typeof(data))**

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

**tempo <- list("LENGTH" = length(data))**

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

**if(all(typeof(data) %in% c("integer", "numeric", "double"))){**

**tempo <- list("RANGE" = range(data[ ! is.infinite(data)], na.rm = TRUE))**

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

**tempo <- list("SUM" = sum(data[ ! is.infinite(data)], na.rm = TRUE))**

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

**tempo <- list("MEAN" = mean(data[ ! is.infinite(data)], na.rm = TRUE))**

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

**tempo <- list("NA.NB" = sum(is.na(data)))**

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

**tempo <- list("INF.NB" = sum(is.infinite(data)))**

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

**}**

**tempo <- list("HEAD" = head(data))**

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

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

**tempo <- list("TAIL" = tail(data))**

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

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

**tempo <- list("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" | all(class(data) %in% c("matrix", "array")))){**

**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))** *# str() print automatically, ls.str() not but does not give the order of the data.frame*

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

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

**fun\_head <- function(data1, n = 6, side = "l"){**

*# AIM*

*# as head() but display the left or right head of big 2D objects*

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

*# fun\_check()*

*# ARGUMENTS*

*# data1: any object but more dedicated for matrix, data frame or table*

*# n: as in head() but for for matrix, data frame or table, 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 (only for matrix, data frame or table)*

*# BEWARE: other arguments of head() not used*

*# RETURN*

*# the head*

*# EXAMPLES*

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

*# obs1 = matrix(1:30, ncol = 5, dimnames = list(letters[1:6], LETTERS[1:5])) ; obs1 ; fun\_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(utils::find("fun\_check", mode = "function")) == 0){**

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

**stop(paste0("\n\n================\n\n", tempo.cat, "\n\n================\n\n"), call. = FALSE)** *# == in stop() to be able to add several messages between ==*

**}**

*# end required function checking*

*# argument checking*

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

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

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

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

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

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

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

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

**}**

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

*# end argument checking*

*# main code*

**if( ! (any(class(data1) %in% c("data.frame", "table")) | all(class(data1) %in% c("matrix", "array")))){** *# before R4.0.0, it was ! any(class(data1) %in% c("matrix", "data.frame", "table"))*

**return(head(data1, n))**

**}else{**

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

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

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

*# AIM*

*# as tail() but display the left or right head of big 2D objects*

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

*# fun\_check()*

*# ARGUMENTS*

*# data1: any object but more dedicated for matrix, data frame or table*

*# n: as in tail() but for for matrix, data frame or table, 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 (only for matrix, data frame or table)*

*# BEWARE: other arguments of tail() not used*

*# RETURN*

*# the tail*

*# EXAMPLES*

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

*# obs1 = matrix(1:30, ncol = 5, dimnames = list(letters[1:6], LETTERS[1:5])) ; obs1 ; fun\_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(utils::find("fun\_check", mode = "function")) == 0){**

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

**stop(paste0("\n\n================\n\n", tempo.cat, "\n\n================\n\n"), call. = FALSE)** *# == in stop() to be able to add several messages between ==*

**}**

*# end required function checking*

*# argument checking*

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

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

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

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

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

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

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

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

**}**

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

*# end argument checking*

*# main code*

**if( ! (any(class(data1) %in% c("data.frame", "table")) | all(class(data1) %in% c("matrix", "array")))){** *# before R4.0.0, it was ! any(class(data1) %in% c("matrix", "data.frame", "table"))*

**return(tail(data1, n))**

**}else{**

**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\_comp\_1d() #### comparison of two 1D datasets (vectors, factors, 1D tables)

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

**fun\_comp\_1d <- function(data1, data2){**

*# AIM*

*# compare two 1D datasets (vector or factor or 1D table, or 1D matrix or 1D array) 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, or 1D matrix or 1D array*

*# data2: vector or factor or 1D table, or 1D matrix or 1D array*

*# 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? NULL if data1 and data2 have no names*

*# $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. NULL if no identical names*

*# $same.name.pos2: position, in data2, of the elements names identical in data1. NULL if no identical names*

*# $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. NULL if no identical elements*

*# $same.element.pos2: position, in data2, of the elements identical in data1. NULL if no identical elements*

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

*# $same.order: logical. Are all elements in the same order? TRUE or FALSE if elements of data1 and data2 are identical but not necessary in the same order. NULL otherwise (different length for instance)*

*# $order1: order of all elements of data1. NULL if $same.order is FALSE*

*# $order2: order of all elements of data2. NULL if $same.order is FALSE*

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

*# $identical.content: logical. Are content objects identical (identical elements, including order, 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\_comp\_1d(obs1, obs2)*

*# obs1 = 1:5 ; obs2 = 1:5 ; names(obs1) <- LETTERS[1:5] ; fun\_comp\_1d(obs1, obs2)*

*# obs1 = 1:5 ; obs2 = 3:6 ; names(obs1) <- LETTERS[1:5] ; names(obs2) <- LETTERS[1:4] ; fun\_comp\_1d(obs1, obs2)*

*# obs1 = factor(LETTERS[1:5]) ; obs2 = factor(LETTERS[1:5]) ; fun\_comp\_1d(obs1, obs2)*

*# obs1 = factor(LETTERS[1:5]) ; obs2 = factor(LETTERS[10:11]) ; fun\_comp\_1d(obs1, obs2)*

*# obs1 = factor(LETTERS[1:5]) ; obs2 = factor(LETTERS[4:7]) ; fun\_comp\_1d(obs1, obs2)*

*# obs1 = factor(c(LETTERS[1:4], "E")) ; obs2 = factor(c(LETTERS[1:4], "F")) ; fun\_comp\_1d(obs1, obs2)*

*# obs1 = 1:5 ; obs2 = factor(LETTERS[1:5]) ; fun\_comp\_1d(obs1, obs2)*

*# obs1 = 1:5 ; obs2 = 1.1:6.1 ; fun\_comp\_1d(obs1, obs2)*

*# obs1 = as.table(1:5); obs2 = as.table(1:5) ; fun\_comp\_1d(obs1, obs2)*

*# obs1 = as.table(1:5); obs2 = 1:5 ; fun\_comp\_1d(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("ERROR IN ", function.name, ": THE data1 ARGUMENT MUST BE A NON NULL VECTOR, FACTOR OR 1D TABLE")**

**stop(paste0("\n\n================\n\n", tempo.cat, "\n\n================\n\n"), call. = FALSE)** *# == in stop() to be able to add several messages between ==*

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

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

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

**stop(paste0("\n\n================\n\n", tempo.cat, "\n\n================\n\n"), call. = FALSE)** *# == in stop() to be able to add several messages between ==*

**}**

**}**

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

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

**stop(paste0("\n\n================\n\n", tempo.cat, "\n\n================\n\n"), call. = FALSE)** *# == in stop() to be able to add several messages between ==*

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

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

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

**stop(paste0("\n\n================\n\n", tempo.cat, "\n\n================\n\n"), call. = FALSE)** *# == in stop() to be able to add several messages between ==*

**}**

**}**

*# 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 <- FALSE**

**class <- NULL**

**same.length <- FALSE**

**length <- NULL**

**same.levels <- NULL # not FALSE to deal with no factors**

**levels <- NULL**

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

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

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

**common.levels <- NULL**

**same.name <- NULL # not FALSE to deal with absence of name**

**name <- NULL**

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

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

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

**common.names <- NULL**

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

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

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

**common.elements <- NULL**

**same.order <- NULL**

**order1 <- NULL**

**order2 <- NULL**

**identical.object <- FALSE**

**identical.content <- FALSE**

**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**

**same.order <- TRUE**

**order1 <- order(data1)**

**order2 <- order(data2)**

**identical.object <- TRUE**

**identical.content <- TRUE**

**}else{**

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

**same.class <- TRUE**

**class <- class(data1)**

**}**

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

**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 <- TRUE**

**levels <- levels(data1)**

**}else{**

**same.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 <- TRUE**

**name <- names(data1)**

**}else{**

**same.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]))**

**}**

**}**

**names(data1) <- NULL** *# names solved -> to do not be disturbed by names*

**names(data2) <- NULL** *# names solved -> to do not be disturbed by names*

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

**identical.content <- TRUE**

**same.order <- TRUE**

**}else if(identical(sort(data1), sort(data2))){**

**same.order <- FALSE**

**order1 <- order(data1)**

**order2 <- order(data2)**

**}**

**}**

**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, same.order = same.order, order1 = order1, order2 = order2, identical.object = identical.object, identical.content = identical.content)**

**return(output)**

**}**

######## fun\_comp\_2d() #### comparison of two 2D datasets (row & col names, dimensions, etc.)

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

**fun\_comp\_2d <- 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)? NULL if nrow(data1) \* nrow(data2) > 1e10*

*# $same.row.pos1: position, in data1, of the rows identical in data2 (not considering row names). Return "TOO BIG FOR EVALUATION" if nrow(data1) \* nrow(data2) > 1e10*

*# $same.row.pos2: position, in data2, of the rows identical in data1 (not considering row names). Return "TOO BIG FOR EVALUATION" if nrow(data1) \* nrow(data2) > 1e10*

*# $any.id.col: logical. is there identical columns (not considering column names)? NULL if ncol(data1) \* ncol(data2) > 1e10*

*# $same.col.pos1: position in data1 of the cols identical in data2 (not considering column names). Return "TOO BIG FOR EVALUATION" if ncol(data1) \* ncol(data2) > 1e10*

*# $same.col.pos2: position in data2 of the cols identical in data1 (not considering column names). Return "TOO BIG FOR EVALUATION" if ncol(data1) \* ncol(data2) > 1e10*

*# $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\_comp\_2d(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\_comp\_2d(obs1, obs2)*

*# large matrices*

*# obs1 = matrix(1:1e6, ncol = 5, dimnames = list(NULL, LETTERS[1:5])) ; obs2 = matrix(as.integer((1:1e6)+1e6/5), ncol = 5, dimnames = list(NULL, LETTERS[1:5])) ; head(obs1) ; head(obs2) ; fun\_comp\_2d(obs1, obs2)*

*# WARNING: when comparing content (rows, columns, or total), double and integer data are considered as different -> double(1) != integer(1)*

*# obs1 = matrix(1:1e6, ncol = 5, dimnames = list(NULL, LETTERS[1:5])) ; obs2 = matrix((1:1e6)+1e6/5, ncol = 5, dimnames = list(NULL, LETTERS[1:5])) ; head(obs1) ; head(obs2) ; fun\_comp\_2d(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\_comp\_2d(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\_comp\_2d(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*

*# data1 = matrix(1:1e6, ncol = 5, dimnames = list(NULL, LETTERS[1:5])) ; data2 = matrix((1:1e6)+1e6/5, ncol = 5, dimnames = list(NULL, LETTERS[1:5]))*

*# function name*

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

*# end function name*

*# argument checking*

**if( ! (any(class(data1) %in% c("data.frame", "table")) | all(class(data1) %in% c("matrix", "array")))){** *# before R4.0.0, it was ! any(class(data1) %in% c("matrix", "data.frame", "table"))*

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

**stop(paste0("\n\n================\n\n", tempo.cat, "\n\n================\n\n"), call. = FALSE)** *# == in stop() to be able to add several messages between ==*

**}**

**if( ! (any(class(data2) %in% c("data.frame", "table")) | all(class(data2) %in% c("matrix", "array")))){** *# before R4.0.0, it was ! any(class(data2) %in% c("matrix", "data.frame", "table"))*

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

**stop(paste0("\n\n================\n\n", tempo.cat, "\n\n================\n\n"), call. = FALSE)** *# == in stop() to be able to add several messages between ==*

**}**

*# 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("data.frame", "table")) | all(class(data1) %in% c("matrix", "array")))){** *# before R4.0.0, it was ! 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("ERROR IN ", function.name, ": THE data1 ARGUMENT IS A 1D TABLE. USE THE fun\_comp\_1d FUNCTION")**

**stop(paste0("\n\n================\n\n", tempo.cat, "\n\n================\n\n"), call. = FALSE)** *# == in stop() to be able to add several messages between ==*

**}**

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

**tempo.cat <- paste0("ERROR IN ", function.name, ": THE data2 ARGUMENT IS A 1D TABLE. USE THE fun\_comp\_1d FUNCTION")**

**stop(paste0("\n\n================\n\n", tempo.cat, "\n\n================\n\n"), call. = FALSE)** *# == in stop() to be able to add several messages between ==*

**}**

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

**same.class <- FALSE**

**}else if( ! (any(class(data1) %in% c("data.frame", "table")) | all(class(data1) %in% c("matrix", "array")))){** *# before R4.0.0, it was ! any(class(data1) %in% c("matrix", "data.frame", "table"))*

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

**stop(paste0("\n\n================\n\n", tempo.cat, "\n\n================\n\n"), call. = FALSE)** *# == in stop() to be able to add several messages between ==*

**}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) %in% c("matrix", "array"))){**

**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) %in% c("matrix", "array"))){**

**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*

**if(all(sapply(data1, FUN = typeof) == "integer") & all(sapply(data2, FUN = typeof) == "integer") & as.double(nrow(data1)) \* nrow(data2) <= 1e10){** *# as.double(nrow(data1)) to prevent integer overflow because R is 32 bits for integers*

**same.row.pos1 <- which(c(as.data.frame(t(data1), stringsAsFactors = FALSE)) %in% c(as.data.frame(t(data2), stringsAsFactors = FALSE)))** *# this work fast with only integers (because 32 bits)*

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

**}else if(as.double(nrow(data1)) \* nrow(data2) <= 1e6){** *# as.double(nrow(data1)) to prevent integer overflow because R is 32 bits for integers*

**same.row.pos1 <- logical(length = nrow(data1))** *# FALSE by default*

**same.row.pos1[] <- FALSE** *# security*

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

**for(i4 in 1:nrow(data2)){**

**same.row.pos1[i3] <- identical(data1[i3, ], data2[i4, ])**

**}**

**}**

**same.row.pos1 <- which(same.row.pos1)**

**same.row.pos2 <- logical(length = nrow(data2))** *# FALSE by default*

**same.row.pos2[] <- FALSE** *# security*

**for(i3 in 1:nrow(data2)){**

**for(i4 in 1:nrow(data1)){**

**same.row.pos2[i3] <- identical(data2[i3, ], data1[i4, ])**

**}**

**}**

**same.row.pos2 <- which(same.row.pos2)**

**}else{**

**same.row.pos1 <- "TOO BIG FOR EVALUATION"**

**same.row.pos2 <- "TOO BIG FOR EVALUATION"**

**}**

**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 if(length(same.row.pos1) == 0 & length(same.row.pos2) == 0){**

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

**}else if(all(same.row.pos1 == "TOO BIG FOR EVALUATION") & all(same.row.pos2 == "TOO BIG FOR EVALUATION")){**

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

**}**

**}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*

**if(all(sapply(data1, FUN = typeof) == "integer") & all(sapply(data2, FUN = typeof) == "integer") & as.double(ncol(data1)) \* ncol(data2) <= 1e10){** *# as.double(ncol(data1)) to prevent integer overflow because R is 32 bits for integers*

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

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

**}else if(as.double(ncol(data1)) \* ncol(data2) <= 1e6){** *# as.double(ncol(data1)) to prevent integer overflow because R is 32 bits for integers*

**same.col.pos1 <- logical(length = ncol(data1))** *# FALSE by default*

**same.col.pos1[] <- FALSE** *# security*

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

**for(i4 in 1:ncol(data2)){**

**same.col.pos1[i3] <- identical(data1[ , i3], data2[ ,i4])**

**}**

**}**

**same.col.pos1 <- which(same.col.pos1)**

**same.col.pos2 <- logical(length = ncol(data2))** *# FALSE by default*

**same.col.pos2[] <- FALSE** *# security*

**for(i3 in 1:ncol(data2)){**

**for(i4 in 1:ncol(data1)){**

**same.col.pos2[i3] <- identical(data2[ , i3], data1[ , i4])**

**}**

**}**

**same.col.pos2 <- which(same.col.pos2)**

**}else{**

**same.col.pos1 <- "TOO BIG FOR EVALUATION"**

**same.col.pos2 <- "TOO BIG FOR EVALUATION"**

**}**

**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 if(length(same.col.pos1) == 0 & length(same.col.pos2) == 0){**

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

**}else if(all(same.col.pos1 == "TOO BIG FOR EVALUATION") & all(same.col.pos2 == "TOO BIG FOR EVALUATION")){**

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

**}**

**}else{**

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

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

**}**

**if(same.dim == TRUE){**

**names(data1) <- NULL**

**row.names(data1) <- NULL**

**names(data2) <- NULL**

**row.names(data2) <- NULL**

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

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

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

**fun\_comp\_list <- 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\_comp\_list(obs1, obs2)*

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

*# obs1 = list(b = 1:5, c = LETTERS[1:2]) ; obs2 = list(LETTERS[5:9], matrix(1:6), 1:5) ; fun\_comp\_list(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("ERROR IN ", function.name, ": THE data1 ARGUMENT MUST BE A LIST")**

**stop(paste0("\n\n================\n\n", tempo.cat, "\n\n================\n\n"), call. = FALSE)** *# == in stop() to be able to add several messages between ==*

**}**

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

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

**stop(paste0("\n\n================\n\n", tempo.cat, "\n\n================\n\n"), call. = FALSE)** *# == in stop() to be able to add several messages between ==*

**}**

*# 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)**

**}**

######## fun\_test() #### test combinations of argument values of a function and return errors (and graphs)

**# add traceback https://stackoverflow.com/questions/47414119/how-to-read-a-traceback-in-r**

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

**fun\_test <- function(fun, arg, val, expect.error = NULL, thread.nb = NULL, print.count = 10, plot.fun = FALSE, export = FALSE, res.path = NULL, lib.path = NULL, cute.path = "C:\\Users\\Gael\\Documents\\Git\_projects\\cute\_little\_R\_functions\\cute\_little\_R\_functions.R"){**

*# AIM*

*# test combinations of argument values of a function*

*# WARNING*

*# Limited to 43 arguments with at least 2 values each. The total number of arguments tested can be more if the additional arguments have a single value. The limit is due to nested "for" loops (https://stat.ethz.ch/pipermail/r-help/2008-March/157341.html), but it should not be a problem since the number of tests would be 2^43 > 8e12*

*# ARGUMENTS*

*# fun: character string indicating the name of the function tested (without brackets)*

*# arg: vector of character strings of arguments of fun. At least arguments that do not have default values must be present in this vector*

*# val: list with number of compartments equal to length of arg, each compartment containing values of the corresponding argument in arg. Each different value must be in a list or in a vector. For instance, argument 3 in arg is a logical argument (values accepted TRUE, FALSE, NA). Thus, compartment 3 of val can be either list(TRUE, FALSE, NA), or c(TRUE, FALSE, NA)*

*# expect.error: list of exactly the same structure as val argument, but containing FALSE or TRUE, depending on whether error is expected (TRUE) or not (FALSE) for each corresponding value of val. A message is returned depending on discrepancies between the expected and observed errors. BEWARE: not always possible to write the expected errors for all the combination of argument values. Ignored if NULL*

*# thread.nb: numeric value indicating the number of available threads. Write NULL if no parallelization wanted*

*# print.count: interger value. Print a working progress message every print.count during loops. BEWARE: can increase substentially the time to complete the process using a small value, like 10 for instance. Use Inf is no loop message desired*

*# plot.fun: logical. Plot the plotting function tested for each test?*

*# export: logical. Export the results into a .RData file and into a .txt file? If FALSE, return a list into the console (see below). BEWARE: will be automatically set to TRUE if thread.nb is not NULL. This means that when using parallelization, the results are systematically exported, not returned into the console*

*# res.path: character string indicating the absolute pathway of folder where the txt results and pdfs, containing all the plots, will be saved. Several txt and pdf, one per thread, if parallelization. Ignored if export is FALSE. Must be specified if thread.nb is not NULL or if export is TRUE*

*# lib.path: character vector specifying the absolute pathways of the directories containing the required packages if not in the default directories. Ignored if NULL*

*# cute.path: character string indicating the absolute path of the cute.R file. Will be remove when cute will be a package. Not considered if thread.nb is NULL*

*# REQUIRED PACKAGES*

*# lubridate*

*# parallel if thread.nb argument is not NULL*

*# if the tested function is in a package, this package must be imported first (no parallelization) or must be in the classical R package folder indicated by the lib.path argument (parallelization)*

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

*# fun\_check()*

*# fun\_get\_message()*

*# fun\_pack()*

*# RETURN*

*# if export is FALSE a list containing:*

*# $fun: the tested function*

*# $data: a data frame of all the combination tested, containing the following columns:*

*# the different values tested, named by arguments*

*# $kind: a vector of character strings indicating the kind of test result: either "ERROR", or "WARNING", or "OK"*

*# $problem: a logical vector indicating if error or not*

*# $expected.error: optional logical vector indicating the expected error specified in the expect.error argument*

*# $message: either NULL if $kind is always "OK", or the messages*

*# $instruction: the initial instruction*

*# $sys.info: system and packages info*

*# if export is TRUE 1) the same list object into a .RData file, 2) also the $data data frame into a .txt file, and 3) if expect.error is non NULL and if any discrepancy, the $data data frame into a .txt file but containing only the rows with discrepancies between expected and observed errors*

*# one or several pdf if a plotting function is tested and if the plot.fun argument is TRUE*

*# EXAMPLES*

*# fun\_test(fun = "unique", arg = c("x", "incomparables"), val = list(x = list(1:10, c(1,1,2,8), NA), incomparable = c(TRUE, FALSE, NA)))*

*# fun\_test(fun = "fun\_round", arg = c("data", "dec.nb", "after.lead.zero"), val = list(L1 = list(c(1, 1.0002256, 1.23568), "a", NA), L2 = list(2, c(1,3), NA), L3 = c(TRUE, FALSE, NA)))*

*# fun\_test(fun = "plot", arg = c("x", "y"), val = list(x = list(1:10, 12:13, NA, (1:10)^2), y = list(1:10, NA, NA)), expect.error = list(x = list(FALSE, TRUE, TRUE, FALSE), y = list(FALSE, TRUE, TRUE)), thread.nb = NULL, plot.fun = TRUE, res.path = "C:\\Users\\Gael\\Desktop\\", lib.path = NULL)*

*# fun\_test(fun = "plot", arg = c("x", "y"), val = list(x = list(1:10, 12:13, NA, (1:10)^2), y = list(1:10, NA, NA)), thread.nb = 4, plot.fun = TRUE, res.path = "C:\\Users\\Gael\\Desktop\\", lib.path = "C:\\Program Files\\R\\R-3.6.1\\library\\")*

*# set.seed(1) ; obs1 <- data.frame(Time = c(rnorm(10), rnorm(10) + 2), Group1 = rep(c("G", "H"), each = 10)) ; fun\_test(fun = "fun\_gg\_boxplot", arg = c("data1", "y", "categ"), val = list(L1 = list(L1 = obs1), L2 = list(L1 = "Time"), L3 = list(L1 = "Group1")))*

*# set.seed(1) ; obs1 <- data.frame(Time = c(rnorm(10), rnorm(10) + 2), Group1 = rep(c("G", "H"), each = 10)) ; fun\_test(fun = "fun\_gg\_boxplot", arg = c("data1", "y", "categ"), val = list(L1 = list(obs1), L2 = "Time", L3 = "Group1"), thread.nb = NULL, plot.fun = TRUE, res.path = "C:\\Users\\Gael\\Desktop\\", lib.path = "C:\\Program Files\\R\\R-3.6.1\\library\\")*

*# library(ggplot2) ; fun\_test(fun = "geom\_histogram", arg = c("data", "mapping"), val = list(x = list(data.frame(X = "a")), y = list(ggplot2::aes(x = X))), thread.nb = NULL, plot.fun = TRUE, res.path = "C:\\Users\\Gael\\Desktop\\", lib.path = "C:\\Program Files\\R\\R-3.6.1\\library\\") # BEWARE: ggplot2::geom\_histogram does not work*

*# DEBUGGING*

*# fun = "unique" ; arg = "x" ; val = list(x = list(1:10, c(1,1,2,8), NA)) ; expect.error = list(x = list(FALSE, FALSE, TRUE)) ; thread.nb = NULL ; plot.fun = FALSE ;**export = FALSE ; res.path = "C:\\Users\\Gael\\Desktop\\" ; lib.path = NULL ; print.count = 1 ; cute.path = "C:\\Users\\Gael\\Documents\\Git\_projects\\cute\_little\_R\_functions\\cute\_little\_R\_functions.R" # for function debugging*

*# fun = "unique" ; arg = c("x", "incomparables") ; val = list(x = list(1:10, c(1,1,2,8), NA), incomparable = c(TRUE, FALSE, NA)) ; expect.error = NULL ; thread.nb = 2 ; plot.fun = FALSE ;**export = TRUE ; res.path = "C:\\Users\\Gael\\Desktop\\" ; lib.path = NULL ; print.count = 10 ; cute.path = "C:\\Users\\Gael\\Documents\\Git\_projects\\cute\_little\_R\_functions\\cute\_little\_R\_functions.R" # for function debugging*

*# fun = "plot" ; arg = c("x", "y") ; val = list(x = list(1:10, 12:13, NA), y = list(1:10, NA, NA)) ; expect.error = list(x = list(FALSE, FALSE, TRUE, FALSE), y = list(FALSE, TRUE, TRUE)) ; print.count = 10 ; thread.nb = NULL ; plot.fun = TRUE ; export = TRUE ; res.path = "C:\\Users\\Gael\\Desktop\\" ; lib.path = NULL # for function debugging*

*# set.seed(1) ; obs1 <- data.frame(Time = c(rnorm(10), rnorm(10) + 2), Group1 = rep(c("G", "H"), each = 10)) ; fun = "fun\_gg\_boxplot" ; arg = c("data1", "y", "categ") ; val = list(L1 = list(L1 = obs1), L2 = list(L1 = "Time"), L3 = list(L1 = "Group1")) ; expect.error = NULL ; print.count = 10 ; thread.nb = NULL ; plot.fun = TRUE ; export = TRUE ; res.path = "C:\\Users\\Gael\\Desktop\\" ; lib.path = NULL # for function debugging*

*# function name*

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

**instruction <- match.call()**

*# end function name*

*# required function checking*

**req.function <- c(**

**"fun\_check",**

**"fun\_get\_message",**

**"fun\_pack"**

**)**

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

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

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

**stop(paste0("\n\n================\n\n", tempo.cat, "\n\n================\n\n"), call. = FALSE)** *# == in stop() to be able to add several messages between ==*

**}**

**}**

*# end required function checking*

*# argument primary checking*

*# arg with no default values*

**if(any(missing(fun) | missing(arg) | missing(val))){**

**tempo.cat <- paste0("ERROR IN ", function.name, ": ARGUMENTS fun, arg AND val HAVE NO DEFAULT VALUE AND REQUIRE ONE")**

**stop(paste0("\n\n================\n\n", tempo.cat, "\n\n================\n\n"), call. = FALSE)** *# == in stop() to be able to add several messages between ==*

**}**

*# end arg with no default values*

*# using fun\_check()*

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

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

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

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

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

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

**if(grepl(x = fun, pattern = "()$")){ # remove ()**

**fun <- sub(x = fun, pattern = "()$", replacement = "")**

**}**

**if( ! exists(fun)){**

**tempo.cat <- paste0("ERROR IN ", function.name, ": CHARACTER STRING IN fun ARGUMENT DOES NOT EXIST IN THE R WORKING ENVIRONMENT: ", paste(fun, collapse = "\n"))**

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

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

**}else if( ! all(class(get(fun)) == "function")){**

**tempo.cat <- paste0("ERROR IN ", function.name, ": fun ARGUMENT IS NOT CLASS \"function\" BUT: ", paste(class(get(fun)), collapse = "\n"), "\nCHECK IF ANY CREATED OBJECT WOULD HAVE THE NAME OF THE TESTED FUNCTION")**

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

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

**}**

**}**

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

**if(tempo$problem == FALSE & length(arg) == 0){**

**tempo.cat <- paste0("ERROR IN ", function.name, ": arg ARGUMENT CANNOT BE LENGTH 0")**

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

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

**}**

**tempo <- fun\_check(data = val, class = "list", fun.name = function.name) ; eval(ee)**

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

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

**tempo1 <- fun\_check(data = val[[i2]], class = "vector", na.contain = TRUE, fun.name = function.name)**

**tempo2 <- fun\_check(data = val[[i2]], class = "list", na.contain = TRUE, fun.name = function.name)**

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

**tempo.cat <- paste0("ERROR IN ", function.name, ": COMPARTMENT ", i2, " OF val ARGUMENT MUST BE A VECTOR OR A LIST")**

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

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

**}else if(tempo1$problem == FALSE){** *# vector split into list compartments*

**val[[i2]] <- split(x = val[[i2]], f = 1:length(val[[i2]]))**

**}**

**}**

**}**

**if( ! is.null(expect.error)){**

**tempo <- fun\_check(data = expect.error, class = "list", fun.name = function.name) ; eval(ee)**

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

**for(i3 in 1:length(expect.error)){**

**tempo1 <- fun\_check(data = expect.error[[i3]], class = "vector", mode = "logical", fun.name = function.name)**

**tempo2 <- fun\_check(data = expect.error[[i3]], class = "list", fun.name = function.name)**

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

**tempo.cat <- paste0("ERROR IN ", function.name, ": COMPARTMENT ", i3, " OF expect.error ARGUMENT MUST BE TRUE OR FALSE")**

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

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

**}else if(tempo1$problem == FALSE){** *# vector split into list compartments*

**expect.error[[i3]] <- split(x = expect.error[[i3]], f = 1:length(expect.error[[i3]]))**

**}**

**}**

**}**

**}**

**if( ! is.null(thread.nb)){**

**tempo <- fun\_check(data = thread.nb, typeof = "integer", double.as.integer.allowed = TRUE, neg.values = FALSE, length = 1, fun.name = function.name) ; eval(ee)**

**if(tempo$problem == FALSE & thread.nb < 1){**

**tempo.cat <- paste0("ERROR IN ", function.name, ": thread.nb PARAMETER MUST EQUAL OR GREATER THAN 1: ", thread.nb)**

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

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

**}**

**}**

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

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

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

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

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

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

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

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

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

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

**}**

**}**

**}**

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

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

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

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

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

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

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

**}**

**}**

**}**

**if( ! is.null(thread.nb)){**

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

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

**if( ! file.exists(cute.path)){**

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

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

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

**}**

**}**

**}**

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

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

**}**

*# end using fun\_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\_check\_dev)) # activate this line and use the function (with no arguments left as NULL) to check arguments status and if they have been checked using fun\_check()*

*# end argument primary checking*

*# second round of checking and data preparation*

*# dealing with NA*

**if(any(is.na(fun)) | any(is.na(arg)) | any(is.na(expect.error)) | any(is.na(thread.nb)) | any(is.na(print.count)) | any(is.na(plot.fun)) | any(is.na(export)) | any(is.na(res.path)) | any(is.na(lib.path))){**

**tempo.cat <- paste0("ERROR IN ", function.name, ": NO ARGUMENT EXCEPT val CAN HAVE NA VALUES")**

**stop(paste0("\n\n================\n\n", tempo.cat, "\n\n================\n\n"), call. = FALSE)** *# == in stop() to be able to add several messages between ==*

**}**

*# end dealing with NA*

*# dealing with NULL*

**if(is.null(fun) | is.null(arg) | is.null(val) | is.null(print.count) | is.null(plot.fun) | is.null(export)){**

**tempo.cat <- paste0("ERROR IN ", function.name, ": THESE ARGUMENTS\nfun\narg\nval\nprint.count\nplot.fun\nexport\nCANNOT BE NULL") #problematic arguments are -> put everywhere**

**stop(paste0("\n\n================\n\n", tempo.cat, "\n\n================\n\n"), call. = FALSE)** *# == in stop() to be able to add several messages between ==*

**}**

*# end dealing with NULL*

**if(length(arg) != length(val)){**

**tempo.cat <- paste0("ERROR IN ", function.name, ": LENGTH OF arg ARGUMENT MUST BE IDENTICAL TO LENGTH OF val ARGUMENT:\nHERE IT IS: ", length(arg), " VERSUS ", length(val))**

**stop(paste0("\n\n================\n\n", tempo.cat, "\n\n================\n\n"), call. = FALSE)**

**}**

**args <- names(formals(get(fun)))**

**if( ! all(arg %in% args)){**

**tempo.cat <- paste0("ERROR IN ", function.name, ": SOME OF THE STRINGS IN arg ARE NOT ARGUMENTS OF fun\nfun ARGUMENTS: ", paste(args, collapse = " "),"\nPROBLEMATIC STRINGS IN arg: ", paste(arg[ ! arg %in% args], collapse = " "))**

**stop(paste0("\n\n================\n\n", tempo.cat, "\n\n================\n\n"), call. = FALSE)**

**}**

**if(sum(sapply(val, FUN = length) > 1) > 43){**

**tempo.cat <- paste0("ERROR IN ", function.name, ": CANNOT TEST MORE THAN 43 ARGUMENTS IF THEY ALL HAVE AT LEAST 2 VALUES EACH\nHERE THE NUMBER IS: ", sum(sapply(val, FUN = length) > 1))**

**stop(paste0("\n\n================\n\n", tempo.cat, "\n\n================\n\n"), call. = FALSE)**

**}**

**if( ! is.null(expect.error)){**

**if(length(val) != length(expect.error)){**

**tempo.cat <- paste0("ERROR IN ", function.name, ": LENGTH OF val ARGUMENT MUST BE IDENTICAL TO LENGTH OF expect.error ARGUMENT:\nHERE IT IS: ", length(val), " VERSUS ", length(expect.error))**

**stop(paste0("\n\n================\n\n", tempo.cat, "\n\n================\n\n"), call. = FALSE)**

**}**

**}**

**if( ! is.null(thread.nb) & is.null(res.path)){**

**tempo.cat <- paste0("ERROR IN ", function.name, ": res.path ARGUMENT MUST BE SPECIFIED IF thread.nb ARGUMENT IS NOT NULL")**

**stop(paste0("\n\n================\n\n", tempo.cat, "\n\n================\n\n"), call. = FALSE)**

**}**

**if(is.null(res.path) & export == TRUE){**

**tempo.cat <- paste0("ERROR IN ", function.name, ": res.path ARGUMENT MUST BE SPECIFIED IF export ARGUMENT TRUE")**

**stop(paste0("\n\n================\n\n", tempo.cat, "\n\n================\n\n"), call. = FALSE)**

**}**

**if( ! is.null(thread.nb) & export == FALSE){**

**export <- TRUE**

**tempo.cat <- paste0("WARNING FROM ", function.name, ": export ARGUMENT CONVERTED TO TRUE BECAUSE thread.nb ARGUMENT IS NOT NULL")**

**warning(paste0("\n", tempo.cat, "\n"), call. = FALSE)**

**}**

*# end second round of checking and data preparation*

*# package checking*

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

**if( ! is.null(thread.nb)){**

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

**}**

*# end package checking*

*# declaration of special plot functions*

**sp.plot.fun <- c("fun\_gg\_scatter", "fun\_gg\_bar", "fun\_gg\_boxplot")**

*# end declaration of special plot functions*

*# main code*

**cat("\nfun\_test JOB IGNITION\n")**

**ini.date <- Sys.time()**

**ini.time <- as.numeric(ini.date)** *# time of process begin, converted into seconds*

**if(export == TRUE){**

**res.path <- paste0(res.path, "/fun\_test\_res\_", trunc(ini.time))**

**if(dir.exists(res.path)){**

**tempo.cat <- paste0("ERROR IN ", function.name, ": FOLDER ALREADY EXISTS\n", res.path, "\nPLEASE RERUN ONCE")**

**stop(paste0("\n\n================\n\n", tempo.cat, "\n\n================\n\n"), call. = FALSE)** *# == in stop() to be able to add several messages between ==*

**}else{**

**dir.create(res.path)**

**}**

**}**

**total.comp.nb <- prod(sapply(val, FUN = "length"))**

**cat(paste0("\nTHE TOTAL NUMBER OF TESTS IS: ", total.comp.nb, "\n"))**

*# creation of the txt instruction that includes several loops*

**loop.string <- NULL**

**end.loop.string <- NULL**

**fun.args <- NULL**

**fun.args2 <- NULL**

**error.values <- NULL**

**arg.values <- "list("**

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

**if(is.null(thread.nb)){**

**if(length(val[[i1]]) > 1){** *# loop only if more than one value in length(val[[i1]])*

**loop.string <- paste0(loop.string, "for(i", i1, " in 1:", length(val[[i1]]), "){")**

**end.loop.string <- paste0(end.loop.string, "}")**

**}**

**}else{**

**loop.string <- "for(i in x){"**

**end.loop.string <- "}"**

**}**

**fun.args <- paste0(**

**fun.args,**

**ifelse(i1 == 1, "", ", "),**

**arg[i1],**

**" = val[[",**

**i1,**

**"]][[",**

**if(is.null(thread.nb)){**

**if(length(val[[i1]]) > 1){**

**paste0("i", i1)**

**}else{**

**"1" # a unique element in val[[i1]]**

**}**

**}else{**

**paste0("i.list[[", i1, "]][i]")**

**},**

**"]]"**

**)**

**fun.args2 <- paste0(**

**fun.args2,**

**ifelse(i1 == 1, "", ", "),**

**arg[i1],**

**" = val[[",**

**i1,**

**"]][[', ",**

**if(is.null(thread.nb)){**

**if(length(val[[i1]]) > 1){**

**paste0("i", i1)**

**}else{**

**"1" # a unique element in val[[i1]]**

**}**

**}else{**

**paste0("i.list[[", i1, "]][i]")**

**},**

**", ']]"**

**)**

**arg.values <- paste0(**

**arg.values,**

**"val[[", i1, "]][[",**

**if(is.null(thread.nb)){**

**if(length(val[[i1]]) > 1){**

**paste0("i", i1)**

**}else{**

**"1" # a unique element in val[[i1]]**

**}**

**}else{**

**paste0("i.list[[", i1, "]][i]")**

**},**

**"]]",**

**ifelse(i1 == length(arg), "", ", ")**

**)**

**error.values <- paste0(**

**error.values,**

**ifelse(i1 == 1, "", " | "),**

**"expect.error[[", i1, "]][[",**

**if(is.null(thread.nb)){**

**if(length(expect.error[[i1]]) > 1){**

**paste0("i", i1)**

**}else{**

**"1" # a unique element in expect.error[[i1]]**

**}**

**}else{**

**paste0("i.list[[", i1, "]][i]")**

**},**

**"]]"**

**)**

**}**

**arg.values <- paste0(arg.values, ")")**

**fun.test <- paste0(fun, "(", fun.args, ")")**

**fun.test2 <- paste0("paste0('", fun, "(", fun.args2, ")')")**

*# plot title for special plot functions*

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

**plot.kind <- "classic"**

**if(fun %in% sp.plot.fun){**

**plot.kind <- "special"**

**if(any(arg %in% "title")){** *# this is for the special functions*

**tempo.match <- regmatches(x = fun.test, m = regexpr(text = fun.test, pattern = "title = .+[,)]"))**

**tempo.match <- substring(tempo.match , 1, nchar(tempo.match) - 1)**

**fun.test <- sub(x = fun.test, pattern = tempo.match, replacement = paste0(tempo.match, "\ntempo.title"))**

**}else{**

**fun.test <- sub(x = fun.test, pattern = ")$", replacement = ", title = tempo.title)")**

**}**

**}**

**}**

*# end plot title for special plot functions*

**kind <- character()**

**problem <- logical()**

**expected.error <- logical()**

**res <- character()**

**count <- 0**

**print.count.loop <- 0**

**plot.count <- 0**

**if(length(arg) == 1){**

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

**}else{** *# length(arg) == 0 already tested above*

**data <- data.frame(t(vector("character", length(arg))), stringsAsFactors = FALSE)[-1, ]** *# -1 to remove the single row created and to have an empty data frame with length(arg) columns*

**}**

**code <- paste(**

**loop.string, '**

**count <- count + 1**

**print.count.loop <- print.count.loop + 1**

**data <- rbind(data, as.character(sapply(eval(parse(text = arg.values)), FUN = "paste", collapse = " ")), stringsAsFactors = FALSE) # each colum is a test**

**tempo.try.error <- fun\_get\_message(data = eval(parse(text = fun.test2)), kind = "error", header = FALSE, env = get(env.name))** *# data argument needs a character string but eval(parse(text = fun.test2)) provides it (eval parse replace the i1, i2, etc., by the correct values, meaning that only val is required in the env.name environment)*

**tempo.try.warning <- fun\_get\_message(data = eval(parse(text = fun.test2)), kind = "warning", header = FALSE, env = get(env.name), print.no = TRUE)** *# data argument needs a character string but eval(parse(text = fun.test2)) provides it (eval parse replace the i1, i2, etc., by the correct values, meaning that only val is required in the env.name environment)*

**if( ! is.null(expect.error)){**

**expected.error <- c(expected.error, eval(parse(text = error.values)))**

**}**

**if( ! is.null(tempo.try.error)){**

**kind <- c(kind, "ERROR")**

**problem <- c(problem, TRUE)**

**res <- c(res, tempo.try.error)**

**}else{**

**if( ! is.null(tempo.try.warning)){**

**kind <- c(kind, "WARNING")**

**problem <- c(problem, FALSE)**

**res <- c(res, tempo.try.warning)**

**}else{**

**kind <- c(kind, "OK")**

**problem <- c(problem, FALSE)**

**res <- c(res, "")**

**}**

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

**invisible(dev.set(window.nb))**

**plot.count <- plot.count + 1**

**tempo.title <- paste0("test\_", sprintf(paste0("%0", nchar(total.comp.nb), "d"), ifelse(is.null(thread.nb), count, x[count])))**

**if(plot.kind == "classic"){**

**eval(parse(text = fun.test))**

**tempo <- fun\_post\_plot(corner.text = tempo.title)**

**}else if(plot.kind == "special"){**

**eval(parse(text = fun.test))**

**}else{**

**tempo.cat <- paste0("INTERNAL CODE ERROR 1 IN ", function.name, ": CODE HAS TO BE MODIFIED")**

**stop(paste0("\n\n================\n\n", tempo.cat, "\n\n================\n\n"), call. = FALSE)** *# == in stop() to be able to add several messages between ==*

**}**

**}**

**}**

**if(print.count.loop == print.count){**

**print.count.loop <- 0**

**tempo.time <- as.numeric(Sys.time())**

**tempo.lapse <- round(lubridate::seconds\_to\_period(tempo.time - ini.time))**

**final.loop <- (tempo.time - ini.time) / count \* ifelse(is.null(thread.nb), total.comp.nb, length(x))** *# expected duration in seconds # intra nb.compar loop lapse: time lapse / cycles done \* cycles remaining*

**final.exp <- as.POSIXct(final.loop, origin = ini.date)**

**cat(paste0(ifelse(is.null(thread.nb), "\n", paste0("\nIN PROCESS ", process.id, " | ")), "LOOP ", format(count, big.mark=","), " / ", format(ifelse(is.null(thread.nb), total.comp.nb, length(x)), big.mark=","), " | TIME SPENT: ", tempo.lapse, " | EXPECTED END: ", final.exp))**

**}**

**if(count == ifelse(is.null(thread.nb), total.comp.nb, length(x))){**

**tempo.time <- as.numeric(Sys.time())**

**tempo.lapse <- round(lubridate::seconds\_to\_period(tempo.time - ini.time))**

**cat(paste0(ifelse(is.null(thread.nb), "\nLOOP PROCESS ENDED | ", paste0("\nPROCESS ", process.id, " ENDED | ")), "LOOP ", format(count, big.mark=","), " / ", format(ifelse(is.null(thread.nb), total.comp.nb, length(x)), big.mark=","), " | TIME SPENT: ", tempo.lapse, "\n\n"))**

**}**

**',**

**end.loop.string**

**)**

*# end creation of the txt instruction that includes several loops*

**if( ! is.null(thread.nb)){**

*# list of i numbers that will be split*

**i.list <- vector("list", length(val)) # positions to split in parallel jobs**

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

**if(i2 == 1){**

**tempo.divisor <- total.comp.nb / length(val[[i2]])**

**i.list[[i2]] <- rep(1:length(val[[i2]]), each = as.integer(tempo.divisor))**

**tempo.multi <- length(val[[i2]])**

**}else{**

**tempo.divisor <- tempo.divisor / length(val[[i2]])**

**i.list[[i2]] <- rep(rep(1:length(val[[i2]]), each = as.integer(tempo.divisor)), time = as.integer(tempo.multi))**

**tempo.multi <- tempo.multi \* length(val[[i2]])**

**}**

**}**

*# end list of i numbers that will be split*

**tempo.cat <- paste0("PARALLELIZATION INITIATED AT: ", ini.date)**

**cat(paste0("\n", tempo.cat, "\n"))**

**tempo.thread.nb = parallel::detectCores(all.tests = FALSE, logical = TRUE)** *# detect the number of threads*

**if(tempo.thread.nb < thread.nb){**

**thread.nb <- tempo.thread.nb**

**}**

**tempo.cat <- paste0("NUMBER OF THREADS USED: ", thread.nb)**

**cat(paste0("\n ", tempo.cat, "\n"))**

**Clust <- parallel::makeCluster(thread.nb, outfile = paste0(res.path, "/fun\_test\_parall\_log.txt"))** *# outfile to print or cat during parallelization (only possible in a file, outfile = "" do not work on windows)*

**tempo.cat <- paste0("SPLIT OF TEST NUMBERS IN PARALLELISATION:")**

**cat(paste0("\n ", tempo.cat, "\n"))**

**cluster.list <- parallel::clusterSplit(Clust, 1:total.comp.nb)** *# split according to the number of cluster*

**str(cluster.list)** *# using print(str()) add a NULL below the result*

**cat("\n")**

**paral.output.list <- parallel::clusterApply(** *# paral.output.list is a list made of thread.nb compartments, each made of n / thread.nb (mat theo column number) compartment. Each compartment receive the corresponding results of fun\_permut(), i.e., data (permuted mat1.perm), warning message, cor (final correlation) and count (number of permutations)*

**cl = Clust,**

**x = cluster.list,**

**function.name = function.name,**

**instruction = instruction,**

**thread.nb = thread.nb,**

**print.count = print.count,**

**total.comp.nb = total.comp.nb,**

**sp.plot.fun = sp.plot.fun,**

**i.list = i.list,**

**fun.tested = fun,**

**arg.values = arg.values,**

**fun.test = fun.test,**

**fun.test2 = fun.test2,**

**kind = kind,**

**problem = problem,**

**res = res,**

**count = count,**

**plot.count = plot.count,**

**data = data,**

**code = code,**

*plot.fun* **=** *plot.fun***,**

*res.path* **=** *res.path***,**

*lib.path* **=** *lib.path***,**

**cute.path = cute.path,**

**fun = function(**

**x,**

**function.name,**

**instruction,**

**thread.nb,**

**print.count,**

**total.comp.nb,**

**sp.plot.fun,**

**i.list,**

**fun.tested,**

**arg.values,**

**fun.test,**

**fun.test2,**

**kind,**

**problem,**

**res,**

**count,**

**plot.count,**

**data,**

**code,**

**plot.fun,**

**res.path,**

**lib.path,**

**cute.path**

**){**

*# check again: very important because another R*

**process.id <- Sys.getpid()**

**cat(paste0("\nPROCESS ID ", process.id, " -> TESTS ", x[1], " TO ", x[length(x)], "\n"))**

**source(cute.path, local = .GlobalEnv)**

**fun\_pack(req.package = "lubridate", lib.path = lib.path, load = TRUE)** *# load = TRUE to be sure that functions are present in the environment. And this prevent to use R.lib.path argument of fun\_python\_pack()*

*# end check again: very important because another R*

*# plot management*

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

**pdf(file = paste0(res.path, "/plots\_from\_fun\_test\_", x[1], ifelse(length(x) == 1, ".pdf", paste0("-", x[length(x)], ".pdf"))))**

**}else{**

**pdf(file = NULL)** *# send plots into a NULL file, no pdf file created*

**}**

**window.nb <- dev.cur()**

**invisible(dev.set(window.nb))**

*# end plot management*

*# new environment*

**env.name <- paste0("env", ini.time)**

**if(exists(env.name, where = -1)){**

**tempo.cat <- paste0("ERROR IN ", function.name, ": ENVIRONMENT env.name ALREADY EXISTS. PLEASE RERUN ONCE")**

**stop(paste0("\n\n================\n\n", tempo.cat, "\n\n================\n\n"), call. = FALSE)** *# == in stop() to be able to add several messages between ==*

**}else{**

**assign(env.name, new.env())**

**assign("val", val, envir = get(env.name)) # var replaced by val**

**}**

*# end new environment*

**ini.date <- Sys.time()**

**ini.time <- as.numeric(ini.date)** *# time of process begin, converted into*

**print.count.loop <- 0**

**suppressMessages(suppressWarnings(eval(parse(text = code))))**

**colnames(data) <- arg**

**if( ! is.null(expect.error)){**

**data <- data.frame(data, kind = kind, problem = problem, expected.error = expected.error, message = res, stringsAsFactors = FALSE)**

**}else{**

**data <- data.frame(data, kind = kind, problem = problem, message = res, stringsAsFactors = FALSE)**

**}**

**row.names(data) <- paste0("test\_", sprintf(paste0("%0", nchar(total.comp.nb), "d"), x))**

**sys.info <- sessionInfo()**

**sys.info$loadedOnly <- sys.info$loadedOnly[order(names(sys.info$loadedOnly))] # sort the packages**

**invisible(dev.off(window.nb))**

**rm(env.name)** *# optional, because should disappear at the end of the function execution*

*# output*

**output <- list(fun = fun, data = data, instruction = instruction, sys.info = sys.info)**

**save(output, file = paste0(res.path, "/fun\_test\_", x[1], ifelse(length(x) == 1, ".RData", paste0("-", x[length(x)], ".RData"))))**

**if(plot.fun == TRUE & plot.count == 0){**

**warning(paste0("\nWARNING FROM ", function.name, " IN PROCESS ", process.id, ": NO PDF PLOT BECAUSE ONLY ERRORS REPORTED\n"), call. = FALSE)**

**file.remove(paste0(res.path, "/plots\_from\_fun\_test\_", x[1], ifelse(length(x) == 1, ".pdf", paste0("-", x[length(x)], ".pdf"))))**

**}**

**table.out <- as.matrix(output$data)**

*# table.out[table.out == ""] <- " " # does not work # because otherwise read.table() converts "" into NA*

**table.out <- gsub(table.out, pattern = "\n", replacement = " ")**

**write.table(table.out, file = paste0(res.path, "/table\_from\_fun\_test\_", x[1], ifelse(length(x) == 1, ".txt", paste0("-", x[length(x)], ".txt"))), row.names = TRUE, col.names = NA, append = FALSE, quote = FALSE, sep = "\t", eol = "\n")**

**}**

**)**

**parallel::stopCluster(Clust)**

*# txt files assembly*

**if(length(cluster.list) > 1){**

**for(i2 in 1:length(cluster.list)){**

**tempo.name <- paste0(res.path, "/table\_from\_fun\_test\_", min(cluster.list[[i2]], na.rm = TRUE), ifelse(length(cluster.list[[i2]]) == 1, ".txt", paste0("-", max(cluster.list[[i2]], na.rm = TRUE), ".txt")))**

**tempo <- read.table(file = tempo.name, header = TRUE, stringsAsFactors = FALSE, sep = "\t", row.names = 1, comment.char = "", colClasses = "character")** *# row.names = 1 (1st column) because now read.table() adds a NA in the header if the header starts by a tabulation, comment.char = "" because colors with #, colClasses = "character" otherwise convert "" (from NULL) into NA*

**file.remove(tempo.name)**

**if(i2 == 1){**

**final.file <- tempo**

**}else{**

**final.file <- rbind(final.file, tempo)**

**}**

**}**

**write.table(final.file, file = paste0(res.path, "/table\_from\_fun\_test\_1-", total.comp.nb, ".txt"), row.names = TRUE, col.names = NA, append = FALSE, quote = FALSE, sep = "\t", eol = "\n")**

**if( ! is.null(expect.error)){**

**final.file <- final.file[ ! final.file$problem == final.file$expected.error, ]**

**if(nrow(final.file) == 0){**

**cat(paste0("NO DISCREPANCY BETWEEN EXPECTED AND OBSERVED ERRORS\n\n"))**

**}else{**

**cat(paste0("DISCREPANCIES BETWEEN EXPECTED AND OBSERVED ERRORS (SEE THE discrepancy\_table\_from\_fun\_test\_1-", total.comp.nb, ".txt FILE)\n\n"))**

**write.table(final.file, file = paste0(res.path, "/discrepancy\_table\_from\_fun\_test\_1-", total.comp.nb, ".txt"), row.names = TRUE, col.names = NA, append = FALSE, quote = FALSE, sep = "\t", eol = "\n")**

**}**

**}**

**}**

*# end txt files assembly*

**}else{**

*# plot management*

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

**pdf(file = paste0(res.path, "/plots\_from\_fun\_test\_1", ifelse(total.comp.nb == 1, ".pdf", paste0("-", total.comp.nb, ".pdf"))))**

**}else{**

**pdf(file = NULL)** *# send plots into a NULL file, no pdf file created*

**}**

**window.nb <- dev.cur()**

**invisible(dev.set(window.nb))**

*# end plot management*

*# new environment*

**env.name <- paste0("env", ini.time)**

**if(exists(env.name, where = -1)){**

**tempo.cat <- paste0("ERROR IN ", function.name, ": ENVIRONMENT env.name ALREADY EXISTS. PLEASE RERUN ONCE")**

**stop(paste0("\n\n================\n\n", tempo.cat, "\n\n================\n\n"), call. = FALSE)** *# == in stop() to be able to add several messages between ==*

**}else{**

**assign(env.name, new.env())**

**assign("val", val, envir = get(env.name)) # var replaced by val**

**}**

*# end new environment*

**suppressMessages(suppressWarnings(eval(parse(text = code))))**

**colnames(data) <- arg**

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

**if( ! is.null(expect.error)){**

**data <- data.frame(data, kind = kind, problem = problem, expected.error = expected.error, message = res, stringsAsFactors = FALSE)**

**}else{**

**data <- data.frame(data, kind = kind, problem = problem, message = res, stringsAsFactors = FALSE)**

**}**

**row.names(data) <- paste0("test\_", sprintf(paste0("%0", nchar(total.comp.nb), "d"), 1:total.comp.nb))**

**sys.info <- sessionInfo()**

**sys.info$loadedOnly <- sys.info$loadedOnly[order(names(sys.info$loadedOnly))] # sort the packages**

**invisible(dev.off(window.nb))**

**rm(env.name)** *# optional, because should disappear at the end of the function execution*

*# output*

**output <- list(fun = fun, data = data, instruction = instruction, sys.info = sys.info)**

**if(plot.fun == TRUE & plot.count == 0){**

**warning(paste0("\nWARNING FROM ", function.name, ": NO PDF PLOT BECAUSE ONLY ERRORS REPORTED\n"), call. = FALSE)**

**file.remove(paste0(res.path, "/plots\_from\_fun\_test\_1", ifelse(total.comp.nb == 1, ".pdf", paste0("-", total.comp.nb, ".pdf"))))**

**}**

**if( ! is.null(expect.error)){**

**expect.data <- output$data[ ! output$data$problem == output$data$expected.error, ]**

**if(nrow(expect.data) == 0){**

**cat(paste0("NO DISCREPANCY BETWEEN EXPECTED AND OBSERVED ERRORS\n\n"))**

**}else{**

**cat(paste0("DISCREPANCIES BETWEEN EXPECTED AND OBSERVED ERRORS (SEE THE ", if(export == TRUE){paste0("discrepancy\_table\_from\_fun\_test\_1", ifelse(total.comp.nb == 1, "", paste0("-", total.comp.nb)), ".txt FILE")}else{"$data RESULT"}, ")\n\n"))**

**if(export == TRUE){**

**expect.data <- as.matrix(expect.data)**

**expect.data <- gsub(expect.data, pattern = "\n", replacement = " ")**

**write.table(expect.data, file = paste0(res.path, "/discrepancy\_table\_from\_fun\_test\_1", ifelse(total.comp.nb == 1, ".txt", paste0("-", total.comp.nb, ".txt"))), row.names = TRUE, col.names = NA, append = FALSE, quote = FALSE, sep = "\t", eol = "\n")**

**}**

**}**

**}**

**if(export == TRUE){**

**save(output, file = paste0(res.path, "/fun\_test\_1", ifelse(total.comp.nb == 1, ".RData", paste0("-", total.comp.nb, ".RData"))))**

**table.out <- as.matrix(output$data)**

**table.out <- gsub(table.out, pattern = "\n", replacement = " ")**

**write.table(table.out, file = paste0(res.path, "/table\_from\_fun\_test\_1", ifelse(total.comp.nb == 1, ".txt", paste0("-", total.comp.nb, ".txt"))), row.names = TRUE, col.names = NA, append = FALSE, quote = FALSE, sep = "\t", eol = "\n")**

**}else{**

**return(output)**

**}**

**}**

**end.date <- Sys.time()**

**end.time <- as.numeric(end.date)**

**total.lapse <- round(lubridate::seconds\_to\_period(end.time - ini.time))**

**cat(paste0("fun\_test JOB END\n\nTIME: ", end.date, "\n\nTOTAL TIME LAPSE: ", total.lapse, "\n\n\n"))**

**}**

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

######## fun\_name\_change() #### check a vector of character strings and modify any string if present in another vector

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

**fun\_name\_change <- function(data1, data2, added.string = "\_modif"){**

*# AIM*

*# this function allow to check if a vector of character strings, like column names of a data frame, has elements present in another vector (vector of reserved words or column names of another data frame before merging)*

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

*# fun\_check()*

*# ARGUMENTS*

*# data1: vector of character strings to check and modify*

*# data2: reference vector of character strings*

*# added.string: string added at the end of the modified string in data1 if present in data2*

*# RETURN*

*# a list containing*

*# $data: the modified data1 (in the same order as in the initial data1)*

*# $ini: the initial elements before modification. NULL if no modification*

*# $post: the modified elements in the same order as in ini. NULL if no modification*

*# EXAMPLES*

*# obs1 <- c("A", "B", "C", "D") ; obs2 <- c("A", "C") ; fun\_name\_change(obs1, obs2)*

*# obs1 <- c("A", "B", "C", "C\_modif1", "D") ; obs2 <- c("A", "A\_modif1", "C") ; fun\_name\_change(obs1, obs2) # the function checks that the new names are neither in obs1 nor in obs2 (increment the number after the added string)*

*# DEBUGGING*

*# data1 = c("A", "B", "C", "D") ; data2 <- c("A", "C") ; added.string = "\_modif" # for function debugging*

*# function name*

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

*# end function name*

*# required function checking*

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

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

**stop(paste0("\n\n================\n\n", tempo.cat, "\n\n================\n\n"), call. = FALSE)** *# == in stop() to be able to add several messages between ==*

**}**

*# end required function checking*

*# argument checking*

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

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

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

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

**tempo <- fun\_check(data = data1, class = "vector", mode = "character", fun.name = function.name) ; eval(ee)**

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

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

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

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

**}**

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

*# end argument checking*

*# main code*

**ini <- NULL**

**post <- NULL**

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

**tempo.names <- data1[data1 %in% data2]**

**ini <- NULL**

**post <- NULL**

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

**count <- 0**

**tempo <- tempo.names[i2]**

**while(any(tempo %in% data2) | any(tempo %in% data1)){**

**count <- count + 1**

**tempo <- paste0(tempo.names[i2], "\_modif", count)**

**}**

**data1[data1 %in% tempo.names[i2]] <- paste0(tempo.names[i2], "\_modif", count)**

**if(count != 0){**

**ini <- c(ini, tempo.names[i2])**

**post <- c(post, paste0(tempo.names[i2], "\_modif", count))**

**}**

**}**

**data <- data1**

**}else{**

**data <- data1**

**}**

**output <- list(data = data, ini = ini, post = post)**

**return(output)**

**}**

######## fun\_df\_remod() #### remodeling a data frame to have column name as a qualitative values and vice-versa

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

**fun\_df\_remod <- 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\_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\_df\_remod(obs)*

*# obs <- data.frame(col1 = (1:4)\*10, col2 = 5:8) ; obs ; fun\_df\_remod(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\_df\_remod(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(utils::find("fun\_check", mode = "function")) == 0){**

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

**stop(paste0("\n\n================\n\n", tempo.cat, "\n\n================\n\n"), call. = FALSE)** *# == in stop() to be able to add several messages between ==*

**}**

*# end required function checking*

*# argument checking*

*# argument checking without fun\_check()*

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

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

**stop(paste0("\n\n================\n\n", tempo.cat, "\n\n================\n\n"), call. = FALSE)** *# == in stop() to be able to add several messages between ==*

**}**

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

*# argument checking with fun\_check()*

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

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

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

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

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

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

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

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

**}**

*# end argument checking with fun\_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\_check\_dev)) # activate this line and use the function (with no arguments left as NULL) to check arguments status and if they have been checked using fun\_check()*

*# end argument checking*

*# 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("ERROR 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")**

**stop(paste0("\n\n================\n\n", tempo.cat, "\n\n================\n\n"), call. = FALSE)** *# == in stop() to be able to add several messages between ==*

**}**

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

**tempo.cat <- paste0("ERROR 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")**

**stop(paste0("\n\n================\n\n", tempo.cat, "\n\n================\n\n"), call. = FALSE)** *# == in stop() to be able to add several messages between ==*

**}**

**}else{**

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

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

**stop(paste0("\n\n================\n\n", tempo.cat, "\n\n================\n\n"), call. = FALSE)** *# == in stop() to be able to add several messages between ==*

**}**

**}**

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

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

**stop(paste0("\n\n================\n\n", tempo.cat, "\n\n================\n\n"), call. = FALSE)** *# == in stop() to be able to add several messages between ==*

**}**

**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\_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*

*# WARNINGS*

*# 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*

*# See ?.Machine or https://stackoverflow.com/questions/5173692/how-to-return-number-of-decimal-places-in-r, with the interexting formula: abs(x - round(x)) > .Machine$double.eps^0.5*

*# 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\_check()*

*# RETURN*

*# the modified vector*

*# EXAMPLES*

*# ini.options <- options()$digits ; options(digits = 8) ; cat(fun\_round(data = c(NA, 10, 100.001, 333.0001254, 12312.1235), dec.nb = 2, after.lead.zero = FALSE), "\n\n") ; options(digits = ini.options)*

*# ini.options <- options()$digits ; options(digits = 8) ; cat(fun\_round(data = c(NA, 10, 100.001, 333.0001254, 12312.1235), dec.nb = 2, after.lead.zero = TRUE), "\n\n") ; options(digits = ini.options)*

*# ini.options <- options()$digits ; options(digits = 8) ; cat(fun\_round(data = c(NA, "10", "100.001", "333.0001254", "12312.1235"), dec.nb = 2, after.lead.zero = FALSE), "\n\n") ; options(digits = ini.options)*

*# ini.options <- options()$digits ; options(digits = 8) ; cat(fun\_round(data = c(NA, "10", "100.001", "333.0001254", "12312.1235"), dec.nb = 2, after.lead.zero = TRUE), "\n\n") ; options(digits = ini.options)*

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

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

**stop(paste0("\n\n================\n\n", tempo.cat, "\n\n================\n\n"), call. = FALSE)** *# == in stop() to be able to add several messages between ==*

**}**

*# end required function checking*

*# argument checking*

*# argument checking without fun\_check()*

**if( ! (all(typeof(data) == "character") | all(typeof(data) == "double") | all(typeof(data) == "integer"))){**

**tempo.cat <- paste0("ERROR IN ", function.name, ": data ARGUMENT MUST BE A VECTOR OF NUMBERS (IN NUMERIC OR CHARACTER MODE)")**

**stop(paste0("\n\n================\n\n", tempo.cat, "\n\n================\n\n"), call. = FALSE)** *# == in stop() to be able to add several messages between ==*

**}**

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

*# argument checking with fun\_check()*

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

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

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

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

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

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

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

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

**}**

*# end argument checking with fun\_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\_check\_dev)) # activate this line and use the function (with no arguments left as NULL) to check arguments status and if they have been checked using fun\_check()*

*# end argument checking*

*# 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\_mat\_rotate() #### 90° clockwise matrix rotation

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

**fun\_mat\_rotate <- 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\_check()*

*# ARGUMENTS*

*# data: matrix (matrix class)*

*# RETURN*

*# the modified matrix*

*# EXAMPLES*

*# obs <- matrix(1:10, ncol = 1) ; obs ; fun\_mat\_rotate(obs)*

*# obs <- matrix(LETTERS[1:10], ncol = 5) ; obs ; fun\_mat\_rotate(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(utils::find("fun\_check", mode = "function")) == 0){**

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

**stop(paste0("\n\n================\n\n", tempo.cat, "\n\n================\n\n"), call. = FALSE)** *# == in stop() to be able to add several messages between ==*

**}**

*# end required function checking*

*# argument checking*

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

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

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

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

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

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

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

**}**

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

*# end argument checking*

*# main code*

**for (i in 1:ncol(data)){data[,i] <- rev(data[,i])}**

**data <- t(data)**

**return(data)**

**}**

######## fun\_mat\_num2color() #### convert a numeric matrix into hexadecimal color matrix

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

**fun\_mat\_num2color <- 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\_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\_mat\_num2color(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(utils::find("fun\_check", mode = "function")) == 0){**

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

**stop(paste0("\n\n================\n\n", tempo.cat, "\n\n================\n\n"), call. = FALSE)** *# == in stop() to be able to add several messages between ==*

**}**

*# end required function checking*

*# argument checking*

*# argument checking with fun\_check()*

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

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

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

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

**tempo <- fun\_check(data = mat1, mode = "numeric", class = "matrix", na.contain = TRUE, neg.values = FALSE, fun.name = function.name) ; eval(ee)**

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

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

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

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

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

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

**}**

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

*# argument checking without fun\_check()*

**if(mat.hsv.h == TRUE & fun\_check(data = mat1, mode = "numeric", prop = TRUE)$problem == TRUE){**

**tempo.cat <- paste0("ERROR IN ", function.name, ": mat1 ARGUMENT MUST BE A MATRIX OF PROPORTIONS SINCE THE mat.hsv.h ARGUMENT IS SET TO TRUE")**

**stop(paste0("\n\n================\n\n", tempo.cat, "\n\n================\n\n"), call. = FALSE)** *# == in stop() to be able to add several messages between ==*

**}**

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

**tempo <- fun\_check(data = forced.color, class = "character")**

**if(any(tempo$problem == TRUE)){**

**paste0("\n\n================\n\n", paste(tempo$text[tempo$problem], collapse = "\n"))**

**stop(paste0("\n\n================\n\n", tempo.cat, "\n\n================\n\n"), call. = FALSE)** *# == in stop() to be able to add several messages between ==*

**}**

**if( ! all(forced.color %in% colors() | grepl(pattern = "^#", forced.color))){** *# check that all strings of forced.color start by #*

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

**stop(paste0("\n\n================\n\n", tempo.cat, "\n\n================\n\n"), call. = FALSE)** *# == in stop() to be able to add several messages between ==*

**}**

**}**

*# end argument checking without fun\_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\_check\_dev)) # activate this line and use the function (with no arguments left as NULL) to check arguments status and if they have been checked using fun\_check()*

*# end argument checking*

*# 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("ERROR IN ", function.name, ": mat1 MUST BE MADE OF VALUES BETWEEN 0 AND 1 BECAUSE mat.hsv.h ARGUMENT SET TO TRUE")**

**stop(paste0("\n\n================\n\n", tempo.cat, "\n\n================\n\n"), call. = FALSE)** *# == in stop() to be able to add several messages between ==*

**}**

**}else{**

**if(any(mat1 - floor(mat1) > 0, na.rm = TRUE) | any(mat1 == 0, na.rm = TRUE)){** *# no need of isTRUE(all.equal()) because we do not require approx here but strictly 0, thus == is ok*

**tempo.cat <- paste0("ERROR IN ", function.name, ": mat1 MUST BE MADE OF INTEGER VALUES WITHOUT 0 BECAUSE mat.hsv.h ARGUMENT SET TO FALSE")**

**stop(paste0("\n\n================\n\n", tempo.cat, "\n\n================\n\n"), call. = FALSE)** *# == in stop() to be able to add several messages between ==*

**}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("ERROR IN ", function.name, ": DUPLICATED VALUES AFTER USING notch (", paste(tempo.different.color[duplicated(tempo.different.color)], collapse = " "), "). TRY ANOTHER notch VALUE")**

**stop(paste0("\n\n================\n\n", tempo.cat, "\n\n================\n\n"), call. = FALSE)** *# == in stop() to be able to add several messages between ==*

**}else if(length(different.color) != length(tempo.different.color)){**

**tempo.cat <- paste0("ERROR IN ", function.name, ": LENGTH OF different.color (", paste(different.color, collapse = " "), ") DIFFERENT FROM LENGTH OF tempo.different.color (", paste(tempo.different.color, collapse = " "), ")")**

**stop(paste0("\n\n================\n\n", tempo.cat, "\n\n================\n\n"), call. = FALSE)** *# == in stop() to be able to add several messages between ==*

**}else{**

**for(i in 1:length(different.color)){**

**mat1[mat1 == different.color[i]] <- tempo.different.color[i]** *# no need of isTRUE(all.equal()) because different.color comes from mat1*

**}**

**}**

**}**

**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)){***# no need of isTRUE(all.equal()) because character*

**tempo.cat <- paste0("ERROR 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 = " "))**

**stop(paste0("\n\n================\n\n", tempo.cat, "\n\n================\n\n"), call. = FALSE)** *# == in stop() to be able to add several messages between ==*

**}else{**

**mat1[which(mat1 == hexa.values.to.change[i])] <- forced.color[i]** *# no need of isTRUE(all.equal()) because character*

**}**

**}**

**}**

**output <- list(mat1.name = mat1.name, colored.mat = mat1, problem = problem, text.problem = text.problem)**

**return(output)**

**}**

######## fun\_mat\_op() #### assemble several matrices with operation

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

**fun\_mat\_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\_check()*

*# fun\_comp\_2d()*

*# 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\_mat\_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\_mat\_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\_mat\_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\_mat\_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(utils::find("fun\_check", mode = "function")) == 0){**

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

**stop(paste0("\n\n================\n\n", tempo.cat, "\n\n================\n\n"), call. = FALSE)** *# == in stop() to be able to add several messages between ==*

**}**

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

**tempo.cat <- paste0("ERROR IN ", function.name, ": REQUIRED fun\_comp\_2d() FUNCTION IS MISSING IN THE R ENVIRONMENT")**

**stop(paste0("\n\n================\n\n", tempo.cat, "\n\n================\n\n"), call. = FALSE)** *# == in stop() to be able to add several messages between ==*

**}**

*# end required function checking*

*# argument checking*

*# argument checking with fun\_check()*

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

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

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

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

**tempo <- fun\_check(data = mat.list, class = "list", fun.name = function.name) ; eval(ee)**

**tempo <- fun\_check(data = kind.of.operation, options = c("+", "-", "\*"), length = 1, fun.name = function.name) ; eval(ee)**

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

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

**}**

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

*# argument checking without fun\_check()*

**if(length(mat.list) < 2){**

**tempo.cat <- paste0("ERROR IN ", function.name, ": mat.list ARGUMENT MUST BE A LIST CONTAINING AT LEAST 2 MATRICES")**

**stop(paste0("\n\n================\n\n", tempo.cat, "\n\n================\n\n"), call. = FALSE)** *# == in stop() to be able to add several messages between ==*

**}**

**for(i1 in 1:length(mat.list)){**

**tempo <- fun\_check(data = mat.list[[i1]], class = "matrix", mode = "numeric", na.contain = TRUE)**

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

**tempo.cat <- paste0("ERROR IN ", function.name, ": ELEMENT ", i1, " OF mat.list ARGUMENT MUST BE A NUMERIC MATRIX")**

**stop(paste0("\n\n================\n\n", tempo.cat, "\n\n================\n\n"), call. = FALSE)** *# == in stop() to be able to add several messages between ==*

**}**

**}**

**ident.row.names <- TRUE**

**ident.col.names <- TRUE**

**for(i1 in 2:length(mat.list)){**

**tempo <- fun\_comp\_2d(data1 = mat.list[[1]], data2 = mat.list[[i1]])**

**if(tempo$same.dim == FALSE){**

**tempo.cat <- paste0("ERROR IN ", function.name, ": MATRIX ", i1, " OF mat.list ARGUMENT MUST HAVE THE SAME DIMENSION (", paste(dim(mat.list[[i1]]), collapse = " "), ") THAN THE MATRIX 1 IN mat.list (", paste(dim(mat.list[[1]]), collapse = " "), ")")**

**stop(paste0("\n\n================\n\n", tempo.cat, "\n\n================\n\n"), call. = FALSE)** *# == in stop() to be able to add several messages between ==*

**}**

**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\_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\_check\_dev)) # activate this line and use the function (with no arguments left as NULL) to check arguments status and if they have been checked using fun\_check()*

*# end argument checking*

*# main code*

**output <- mat.list[[1]]**

**for(i1 in 2:length(mat.list)){**

**output <- get(kind.of.operation)(output, mat.list[[i1]])**

**}**

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

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

**stop(paste0("\n\n================\n\n", tempo.cat, "\n\n================\n\n"), call. = FALSE)** *# == in stop() to be able to add several messages between ==*

**}**

*# end required function checking*

*# argument checking*

*# argument checking with fun\_check()*

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

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

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

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

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

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

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

**}**

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

*# argument checking without fun\_check()*

**if(ncol(mat) != nrow(mat)){**

**tempo.cat <- paste0("ERROR IN ", function.name, ": mat ARGUMENT MUST BE A SQUARE MATRIX")**

**stop(paste0("\n\n================\n\n", tempo.cat, "\n\n================\n\n"), call. = FALSE)** *# == in stop() to be able to add several messages between ==*

**}**

**if(any(mat %in% c(Inf, -Inf, NA))){**

**tempo.cat <- paste0("ERROR IN ", function.name, ": mat ARGUMENT MUST BE A MATRIX WITHOUT Inf, -Inf OR NA")**

**stop(paste0("\n\n================\n\n", tempo.cat, "\n\n================\n\n"), call. = FALSE)** *# == in stop() to be able to add several messages between ==*

**}**

**if(all(mat == 0) & ncol(mat) == 1){**

**tempo.cat <- paste0("ERROR IN ", function.name, ": mat ARGUMENT CANNOT BE A SQUARE MATRIX MADE OF A SINGLE CASE OF 0")**

**stop(paste0("\n\n================\n\n", tempo.cat, "\n\n================\n\n"), call. = FALSE)** *# == in stop() to be able to add several messages between ==*

**}**

*# end argument checking without fun\_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\_check\_dev)) # activate this line and use the function (with no arguments left as NULL) to check arguments status and if they have been checked using fun\_check()*

*# end argument checking*

*# 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, warn.print = FALSE){**

*# 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*

*# WARNINGS*

*# a plot verification using fun\_gg\_heatmap() is recommanded*

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

*# fun\_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*

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

*# RETURN*

*# a list containing:*

*# $mat: the filled matrix*

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

*# 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, warn.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, warn.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, warn.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", warn.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, warn.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, warn.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 ; warn.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 ; warn.print = TRUE # for function debugging # topleft example*

*# mat = matrix(c(0,0,0,2, 0,0,3,0, 0,3,0,NA, 5,0,0,0), ncol = 4) ; empty.cell.string = NA ; warn.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(utils::find("fun\_check", mode = "function")) == 0){**

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

**stop(paste0("\n\n================\n\n", tempo.cat, "\n\n================\n\n"), call. = FALSE)** *# == in stop() to be able to add several messages between ==*

**}**

*# end required function checking*

*# argument checking*

*# argument checking with fun\_check()*

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

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

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

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

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

**tempo <- fun\_check(data = empty.cell.string, class = "vector", na.contain = TRUE, fun.name = function.name) ; eval(ee)**

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

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

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

**}**

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

*# argument checking without fun\_check()*

**if(ncol(mat) != nrow(mat)){**

**tempo.cat <- paste0("ERROR IN ", function.name, ": mat ARGUMENT MUST BE A SQUARE MATRIX")**

**stop(paste0("\n\n================\n\n", tempo.cat, "\n\n================\n\n"), call. = FALSE)** *# == in stop() to be able to add several messages between ==*

**}**

**if( ! (mode(mat) %in% c("numeric", "character"))){**

**tempo.cat <- paste0("ERROR IN ", function.name, ": mat ARGUMENT MUST BE A NUMERIC OR CHARACTER MATRIX")**

**stop(paste0("\n\n================\n\n", tempo.cat, "\n\n================\n\n"), call. = FALSE)** *# == in stop() to be able to add several messages between ==*

**}**

**if(nrow(mat) == 1 & ncol(mat) == 1){**

**tempo.cat <- paste0("ERROR IN ", function.name, ": mat ARGUMENT CANNOT BE A SQUARE MATRIX MADE OF A SINGLE CASE")**

**stop(paste0("\n\n================\n\n", tempo.cat, "\n\n================\n\n"), call. = FALSE)** *# == in stop() to be able to add several messages between ==*

**}**

**if(ifelse(is.na(empty.cell.string), ! any(is.na(mat)), ! any(mat == empty.cell.string, na.rm = TRUE))){**

**tempo.cat <- paste0("ERROR IN ", function.name, ": mat ARGUMENT MATRIX MUST HAVE CELLS WITH THE EMPTY STRING SPECIFIED IN empty.cell.string ARGUMENT")**

**stop(paste0("\n\n================\n\n", tempo.cat, "\n\n================\n\n"), call. = FALSE)** *# == in stop() to be able to add several messages between ==*

**}**

*# end argument checking without fun\_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\_check\_dev)) # activate this line and use the function (with no arguments left as NULL) to check arguments status and if they have been checked using fun\_check()*

*# end argument checking*

*# main code*

**list.diag <- vector("list", length = nrow(mat) - 1)**

**for(i1 in 1:(nrow(mat) - 1)){**

**list.diag[[i1]] <- numeric(length = nrow(mat) - i1)**

**}**

**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) - i2), (ncol(mat) -i2):1)))]",** *# topleft part*

**"mat[as.matrix(as.data.frame(list(1:(nrow(mat) - i2), (1:ncol(mat))[-(1:i2)])))]",** *# topright part*

**"mat[as.matrix(as.data.frame(list((1 + i2):nrow(mat), ncol(mat):(1 + i2))))]",** *# bottomright part*

**"mat[as.matrix(as.data.frame(list((1 + i2):nrow(mat), 1:(ncol(mat) -i2))))]"** *# bottomleft part*

**)**

*# empty part detection*

**tempo.list.diag <- list.diag**

**empty.sector <- NULL**

**full.sector <- NULL**

**ini.warning.length <- options()$warning.length**

**options(warning.length = 8170)**

**on.exit(exp = options(warning.length = ini.warning.length))**

**warn <- NULL**

**warn.count <- 0**

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

**tempo.list.diag <- list.diag**

**for(i2 in 1:(nrow(mat) - 1)){**

**tempo.list.diag[[i2]] <- eval(parse(text = diag.scan[i1]))**

**if(ifelse(is.na(empty.cell.string), ! all(is.na(tempo.list.diag[[i2]])), ! (all(tempo.list.diag[[i2]] == empty.cell.string, na.rm = TRUE) & ! (is.na(all(tempo.list.diag[[i2]] == empty.cell.string, na.rm = FALSE)))))){** *# I had to add this ! (is.na(all(tempo.list.diag[[i2]] == empty.cell.string, na.rm = FALSE))) because all(tempo.list.diag[[i2]] == empty.cell.string, na.rm = FALSE) gives NA and not FALSE if one NA in tempo.list.diag[[i2]] -> not good for if()*

**full.sector <- c(full.sector, sector[i1])**

**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[i1])**

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

**tempo.warn <- paste0("(", warn.count,") EMPTY SECTOR DETECTED ON THE ", toupper(sector[i1]), " CORNER, FULL OF ", empty.cell.string)**

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

**}else{**

**tempo.cat <- paste0("ERROR IN ", function.name, ": THE ", toupper(sector[i1]), " SECTOR, DETECTED AS EMPTY, IS NOT? DIFFERENT VALUES IN THIS SECTOR:\n", paste(names(table(unlist(tempo.list.diag), useNA = "ifany")), collapse = " "))**

**stop(paste0("\n\n================\n\n", tempo.cat, "\n\n================\n\n", ifelse(is.null(warn), "", paste0("IN ADDITION\nWARNING", ifelse(warn.count > 1, "S", ""), ":\n\n", warn))), call. = FALSE)** *# == in stop() to be able to add several messages between ==*

**}**

**}**

**}**

*# end empty part detection*

**if(length(empty.sector) == 0){**

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

**tempo.warn <- paste0("(", warn.count,") ACCORDING TO empty.cell.string ARGUMENT (", empty.cell.string, "), mat ARGUMENT MATRIX HAS ZERO EMPTY HALF PART")**

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

**}else{**

**if(length(empty.sector) > 1){**

**tempo.cat <- paste0("ERROR 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 = " "))**

**stop(paste0("\n\n================\n\n", tempo.cat, "\n\n================\n\n", ifelse(is.null(warn), "", paste0("IN ADDITION\nWARNING", ifelse(warn.count > 1, "S", ""), ":\n\n", warn))), call. = FALSE)** *# == in stop() to be able to add several messages between ==*

**}else if(any(full.sector %in% empty.sector, na.rm = TRUE)){**

**tempo.cat <- paste0("ERROR 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 = " "))**

**stop(paste0("\n\n================\n\n", tempo.cat, "\n\n================\n\n", ifelse(is.null(warn), "", paste0("IN ADDITION\nWARNING", ifelse(warn.count > 1, "S", ""), ":\n\n", warn))), call. = FALSE)** *# == in stop() to be able to add several messages between ==*

**}else if(length(empty.sector) + length(full.sector)!= 4){**

**tempo.cat <- paste0("ERROR IN ", function.name, ": THE FUNCTION HAS DETECTED MORE OR LESS SECTORS THAN 4:\nHALF SECTORS:", paste(empty.sector, collapse = " "), "\nFULL SECTORS:", paste(full.sector, collapse = " "))**

**stop(paste0("\n\n================\n\n", tempo.cat, "\n\n================\n\n", ifelse(is.null(warn), "", paste0("IN ADDITION\nWARNING", ifelse(warn.count > 1, "S", ""), ":\n\n", warn))), call. = FALSE)** *# == in stop() to be able to add several messages between ==*

**}else{**

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

**tempo.warn <- paste0("(", warn.count,") ", toupper(empty.sector), " SECTOR HAS BEEN COMPLETED TO BECOME SYMMETRICAL")**

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

**}**

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

**warning(paste0("FROM ", function.name, ":\n\n", warn), call. = FALSE)**

**}**

**return(list(mat = mat, warn = warn))**

**}**

######## fun\_permut() #### progressively breaks a vector order

**fun\_permut <- function(data1, data2 = NULL, n = NULL, seed = NULL, print.count = 10, text.print = "", cor.method = "spearman", cor.limit = 0.2, warn.print = FALSE, lib.path = 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.*

*# designed for discrete values, but worls also for continuous values*

*# WARNINGS*

*# see # https://www.r-bloggers.com/strategies-to-speedup-r-code/ for code speedup*

*# the random switch of non consecutive positions (ABCD -> DBCA for instance) does not work very well as the correlation is quickly obtained but the initial vector structure is mainly kept (no much order). Ths code would be: pos <- ini.pos[1:2] ; pos <- sample.int(n = n , size = 2, replace = FALSE) ; tempo.pos[pos] <- tempo.pos[rev(pos)]*

*# 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(). Write NULL if random result is required, an integer otherwise. BEWARE: if not NULL, fun\_permut() will systematically return the same result when the other parameters keep the same settings*

*# print.count: interger value. Print a working progress message every print.count during loops. BEWARE: can increase substentially the time to complete the process using a small value, like 10 for instance. Use Inf is no loop message desired*

*# text.print: optional message to add to the working progress message every print.count 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). Ignored if data2 is not specified. Compute the correlation between data1 and data2, permute the data1 values, and stop the permutation process when the correlation between data1 and data2 decreases down below the cor limit value (0.2 by default). If cor(data1, data2) is negative, then -cor.limit is used and the process stops until the correlation between data1 and data2 increases up over 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 automatically uses -cor.limit. If the initial correlation is already below cor.limit (positive correlation) or over -cor.limit (negative correlation), then the data1 value positions are completely randomized (correlation between data1 and data2 is expected to be 0)*

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

*# lib.path: character vector specifying the absolute pathways of the directories containing the required packages if not in the default directories. Ignored if NULL*

*# REQUIRED PACKAGES*

*# lubridate*

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

*# fun\_check()*

*# fun\_pack()*

*# fun\_round()*

*# RETURN*

*# a list containing:*

*# $data: the modified vector*

*# $warn: potential warning messages (in case of negative correlation when data2 is specified). NULL if non warning message*

*# $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, according to cor.method*

*# $count: the number of loops used*

*# EXAMPLES*

*# example (1) showing that for loop, used in fun\_permut(), is faster than while loop*

*# ini.time <- as.numeric(Sys.time()) ; count <- 0 ; for(i0 in 1:1e9){count <- count + 1} ; tempo.time <- as.numeric(Sys.time()) ; tempo.lapse <- round(lubridate::seconds\_to\_period(tempo.time - ini.time)) ; tempo.lapse*

*# example (2) showing that for loop, used in fun\_permut(), is faster than while loop*

*# ini.time <- as.numeric(Sys.time()) ; count <- 0 ; while(count < 1e9){count <- count + 1} ; tempo.time <- as.numeric(Sys.time()) ; tempo.lapse <- round(lubridate::seconds\_to\_period(tempo.time - ini.time)) ; tempo.lapse*

*# fun\_permut(data1 = LETTERS[1:5], data2 = NULL, n = 100, seed = 1, print.count = 10, text.print = "CPU NB 4")*

*# fun\_permut(data1 = 101:110, data2 = 21:30, seed = 1, print.count = 1e4, text.print = "", cor.method = "spearman", cor.limit = 0.2)*

*# a way to use the cor.limit argument just considering data1*

*# obs1 <- 101:110 ; fun\_permut(data1 = obs1, data2 = obs1, seed = 1, print.count = 10, cor.method = "spearman", cor.limit = 0.2)*

*# fun\_permut(data1 = 1:1e3, data2 = 1e3:1, seed = 1, print.count = 1e6, text.print = "", cor.method = "spearman", cor.limit = 0.7)*

*# fun\_permut(data1 = 1:1e2, data2 = 1e2:1, seed = 1, print.count = 1e3, cor.limit = 0.5)*

*# fun\_permut(data1 = c(0,0,0,0,0), n = 5, data2 = NULL, seed = 1, print.count = 1e3, cor.limit = 0.5)*

*# DEBUGGING*

*# data1 = LETTERS[1:5] ; data2 = NULL ; n = 1e6 ; seed = NULL ; print.count = 1e3 ; text.print = "" ; cor.method = "spearman" ; cor.limit = 0.2 ; warn.print = TRUE ; lib.path = NULL*

*# data1 = LETTERS[1:5] ; data2 = NULL ; n = 10 ; seed = 22 ; print.count = 10 ; text.print = "" ; cor.method = "spearman" ; cor.limit = 0.2 ; warn.print = TRUE ; lib.path = NULL*

*# data1 = 101:110 ; data2 = 21:30 ; n = 10 ; seed = 22 ; print.count = 10 ; text.print = "" ; cor.method = "spearman" ; cor.limit = 0.2 ; warn.print = TRUE ; lib.path = NULL*

*# data1 = 1:1e3 ; data2 = 1e3:1 ; n = 20 ; seed = 22 ; print.count = 1e6 ; text.print = "" ; cor.method = "spearman" ; cor.limit = 0.5 ; warn.print = TRUE ; lib.path = NULL*

*# function name*

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

*# end function name*

*# required function checking*

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

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

**stop(paste0("\n\n================\n\n", tempo.cat, "\n\n================\n\n"), call. = FALSE)** *# == in stop() to be able to add several messages between ==*

**}**

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

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

**stop(paste0("\n\n================\n\n", tempo.cat, "\n\n================\n\n"), call. = FALSE)** *# == in stop() to be able to add several messages between ==*

**}**

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

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

**stop(paste0("\n\n================\n\n", tempo.cat, "\n\n================\n\n"), call. = FALSE)** *# == in stop() to be able to add several messages between ==*

**}**

*# end required function checking*

*# argument checking*

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

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

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

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

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

**if(tempo$problem == FALSE & length(data1) < 2){**

**tempo.cat <- paste0("ERROR IN ", function.name, ": data1 ARGUMENT MUST BE A VECTOR OF MINIMUM LENGTH 2. HERE IT IS: ", length(data1))**

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

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

**}**

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

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

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

**tempo.cat <- paste0("ERROR IN ", function.name, ": data1 MUST BE A NUMERIC VECTOR IF data2 ARGUMENT IS SPECIFIED")**

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

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

**}**

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

**if(length(data1) != length(data2)){**

**tempo.cat <- paste0("ERROR IN ", function.name, ": data1 AND data2 MUST BE VECTOR OF SAME LENGTH. HERE IT IS ", length(data1)," AND ", length(data2))**

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

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

**}**

**}else if(is.null(n)){**

**tempo.cat <- paste0("ERROR IN ", function.name, ": n ARGUMENT CANNOT BE NULL IF data2 ARGUMENT IS NULL")**

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

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

**}**

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

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

**}**

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

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

**}**

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

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

**tempo <- fun\_check(data = cor.method, options = c("pearson", "kendall", "spearman"), length =1, fun.name = function.name) ; eval(ee)**

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

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

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

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

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

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

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

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

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

**}**

**}**

**}**

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

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

**}**

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

*# end argument checking*

*# package checking*

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

*# end package checking*

*# main code*

*# code that protects set.seed() in the global environment*

*# see also Protocol 100-rev0 Parallelization in R.docx*

**if(exists(".Random.seed", envir = .GlobalEnv)){** *# if .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 seeding -> return to complete randomness*

**}**

*# end code that protects set.seed() in the global environment*

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

**set.seed(seed)**

**}**

**ini.date <- Sys.time()** *# time of process begin, converted into seconds*

**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*

**# pos.selec.seq <- ini.pos[-length(data1)]** *# selection of 1 position in initial position, without the last because always up permutation (pos -> pos+1 & pos+1 -> pos)*

**pos.selec.seq.max <- length(ini.pos) - 1** *# max position (used by sample.int() function). See below for - 1*

**ini.warning.length <- options()$warning.length**

**options(warning.length = 8170)**

**on.exit(exp = options(warning.length = ini.warning.length))**

**warn <- NULL**

**warn.count <- 0**

**count <- 0**

**round <- 0**

**BREAK <- FALSE**

**tempo.cor <- 0**

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

**if(length(table(data1)) == 1){**

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

**tempo.warn <- paste0("(", warn.count,") NO PERMUTATION PERFORMED BECAUSE data1 ARGUMENT SEEMS TO BE MADE OF IDENTICAL ELEMENTS: ", names(table(data1)))**

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

**}else{**

**if(print.count > n){**

**print.count <- n**

**}**

**cat(paste0("\n", ifelse(text.print == "", "", paste0(text.print, " | ")), "FOR LOOP OF ", n, " LOOPS INITIATED | LOOP COUNT: ", format(count, big.mark=",")))**

**print.count.loop <- logical(length = print.count)**

**print.count.loop[length(print.count.loop)] <- TRUE** *# not this to avoid long vector, but not forget to reset during printing: print.count.loop[(1:trunc(n / print.count) \* print.count)] <- TRUE # counter to speedup*

**count.loop <- 0**

**pos <- sample.int(n = pos.selec.seq.max , size = print.count, replace = TRUE)** *# selection of random positions. BEWARE: n = pos.selec.seq.max because already - 1 (see above) but is connected to tempo.pos[c(pos2 + 1, pos2)] <- tempo.pos[c(pos2, pos2 + 1)]*

**tempo.date.loop <- Sys.time()**

**tempo.time.loop <- as.numeric(tempo.date.loop)**

**for(i3 in 1:n){**

**count.loop <- count.loop + 1**

**pos2 <- pos[count.loop]** *# selection of 1 position*

**tempo.pos[c(pos2 + 1, pos2)] <- tempo.pos[c(pos2, pos2 + 1)]**

**if(print.count.loop[count.loop]){**

**count.loop <- 0**

**pos <- sample.int(n = pos.selec.seq.max , size = print.count, replace = TRUE)** *# BEWARE: never forget to resample here*

**tempo.time <- as.numeric(Sys.time())**

**tempo.lapse <- round(lubridate::seconds\_to\_period(tempo.time - tempo.time.loop))**

**final.loop <- (tempo.time - tempo.time.loop) / i3 \* n** *# expected duration in seconds*

**final.exp <- as.POSIXct(final.loop, origin = tempo.date.loop)**

**cat(paste0("\n", ifelse(text.print == "", "", paste0(text.print, " | ")), "FOR LOOP ", i3, " / ", n, " | TIME SPENT: ", tempo.lapse, " | EXPECTED END: ", final.exp))**

**}**

**}**

**count <- count + n** *# out of the loop to speedup*

**cat(paste0("\n", ifelse(text.print == "", "", paste0(text.print, " | ")), "FOR LOOP ENDED | LOOP COUNT: ", format(count, big.mark=",")))**

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

**}**

**}else{**

**if(length(table(data1)) == 1){**

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

**tempo.warn <- paste0("(", warn.count,") NO PERMUTATION PERFORMED BECAUSE data1 ARGUMENT SEEMS TO BE MADE OF IDENTICAL ELEMENTS: ", names(table(data1)))**

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

**tempo.cor <- 1**

**}else if(length(table(data2)) == 1){**

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

**tempo.warn <- paste0("(", warn.count,") NO PERMUTATION PERFORMED BECAUSE data2 ARGUMENT SEEMS TO BE MADE OF IDENTICAL ELEMENTS: ", names(table(data2)))**

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

**tempo.cor <- 1**

**}else{**

**cor.ini <- cor(x = data1, y = data2, use = "pairwise.complete.obs", method = cor.method)**

**tempo.cor <- cor.ini** *# correlation that will be modified during loops*

**neg.cor <- FALSE**

**if(tempo.cor < 0){**

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

**tempo.warn <- paste0("(", warn.count,") INITIAL ", toupper(cor.method), " CORRELATION BETWEEN data1 AND data2 HAS BEEN DETECTED AS NEGATIVE: ", tempo.cor, ". THE LOOP STEPS WILL BE PERFORMED USING POSITIVE CORRELATIONS BUT THE FINAL CORRELATION WILL BE NEGATIVE")**

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

**neg.cor <- TRUE**

**tempo.cor <- abs(tempo.cor)**

**cor.ini <- abs(cor.ini)**

**}**

**if(tempo.cor < cor.limit){** *# randomize directly all the position to be close to correlation zero*

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

**tempo.warn <- paste0("(", warn.count,") INITIAL ABSOLUTE VALUE OF THE ", toupper(cor.method), " CORRELATION ", fun\_round(tempo.cor), " BETWEEN data1 AND data2 HAS BEEN DETECTED AS BELOW THE CORRELATION LIMIT PARAMETER ", cor.limit, "\nTHE data1 SEQUENCE HAS BEEN COMPLETELY RANDOMIZED TO CORRESPOND TO CORRELATION ZERO")**

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

**for(i4 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)**

**}**

**count <- count + 5** *# out of the loop to speedup*

**}else{**

*# smallest correlation decrease*

**count <- count + 1** *# 1 and not 0 because already 1 performed just below*

**pos <- sample.int(n = pos.selec.seq.max , size = 1, replace = TRUE)** *# selection of 1 position # pos.selec.seq.max because selection of 1 position in initial position, without the last because always up permutation (pos -> pos+1 & pos+1 -> pos)*

**tempo.pos[c(pos + 1, pos)] <- tempo.pos[c(pos, pos + 1)]**

**tempo.cor <- abs(cor(x = data1[tempo.pos], y = data2, use = "pairwise.complete.obs", method = cor.method))**

**smallest.cor.dec <- cor.ini - tempo.cor**

*# end smallest correlation decrease*

*# going out of tempo.cor == cor.ini*

**cat(paste0("\n", ifelse(text.print == "", "", paste0(text.print, " | ")), "CORRELATION DECREASE AFTER A SINGLE PERMUTATION: ", fun\_round(smallest.cor.dec, 4)))**

**cat(paste0("\n", ifelse(text.print == "", "", paste0(text.print, " | ")), "FIRST WHILE LOOP STEP -> GOING OUT FROM EQUALITY | LOOP COUNT: ", format(count, big.mark=","), " | CORRELATION LIMIT: ", fun\_round(cor.limit, 4), " | ABS TEMPO CORRELATION: ", fun\_round(tempo.cor, 4)))**

**print.count.loop <- logical(length = print.count)**

**print.count.loop[length(print.count.loop)] <- TRUE** *# counter to speedup*

**count.loop <- 0** *#*

**pos <- sample.int(n = pos.selec.seq.max , size = print.count, replace = TRUE)** *# selection of random positions. BEWARE: n = pos.selec.seq.max because already - 1 (see above) but is connected to tempo.pos[c(pos2 + 1, pos2)] <- tempo.pos[c(pos2, pos2 + 1)]*

**tempo.date.loop <- Sys.time()**

**tempo.time.loop <- as.numeric(tempo.date.loop)**

**while(tempo.cor == cor.ini){** *# to be out of equality between tempo.cor and cor.ini at the beginning (only valid for very long vector)*

**count <- count + 1**

**count.loop <- count.loop + 1**

**pos2 <- pos[count.loop]**

**tempo.pos[c(pos2 + 1, pos2)] <- tempo.pos[c(pos2, pos2 + 1)]**

**tempo.cor <- abs(cor(x = data1[tempo.pos], y = data2, use = "pairwise.complete.obs", method = cor.method))**

**if(print.count.loop[count.loop]){**

**count.loop <- 0**

**pos <- sample.int(n = pos.selec.seq.max , size = print.count, replace = TRUE)** *# BEWARE: never forget to resample here*

**tempo.time <- as.numeric(Sys.time())**

**tempo.lapse <- round(lubridate::seconds\_to\_period(tempo.time - tempo.time.loop))**

**cat(paste0("\n", ifelse(text.print == "", "", paste0(text.print, " | ")), "FIRST WHILE LOOP STEP", format(count.loop, big.mark=","), " / ? | COUNT: ", format(count, big.mark=","), " | CORRELATION LIMIT: ", fun\_round(cor.limit, 4), " | ABS TEMPO CORRELATION: ", fun\_round(tempo.cor, 4), " | TIME SPENT: ", tempo.lapse))**

**}**

**}**

**tempo.time <- as.numeric(Sys.time())**

**tempo.lapse <- round(lubridate::seconds\_to\_period(tempo.time - ini.time))**

**cat(paste0("\n", ifelse(text.print == "", "", paste0(text.print, " | ")), "FIRST WHILE LOOP STEP END | LOOP COUNT: ", format(count, big.mark=","), " | CORRELATION LIMIT: ", fun\_round(cor.limit, 4), " | ABS TEMPO CORRELATION: ", fun\_round(tempo.cor, 4), " | TOTAL SPENT TIME: ", tempo.lapse))**

**if(tempo.cor < cor.limit){**

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

**tempo.warn <- paste0("(", warn.count,") THE FIRST FOR & WHILE LOOP STEPS HAVE BEEN TOO FAR AND SUBSEQUENT LOOP STEPS WILL NOT RUN")**

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

**}**

*# end going out of tempo.cor == cor.ini*

*# estimation of the average correlation decrease per loop on x loops and for loop execution*

**cat(paste0("\n", ifelse(text.print == "", "", paste0(text.print, " | ")), "WHILE/FOR LOOPS INITIATION | LOOP COUNT: ", format(count, big.mark=","), " | CORRELATION LIMIT: ", fun\_round(cor.limit, 4), " | ABS TEMPO CORRELATION: ", fun\_round(tempo.cor, 4)))**

**count.est <- 1e5**

**first.round <- TRUE**

**GOBACK <- FALSE**

**while(tempo.cor > cor.limit){**

**round <- round + 1**

*# estimation step*

**if(first.round == TRUE){**

**first.round <- FALSE**

**cor.dec.per.loop <- numeric(length = 5)**

**loop.nb.est <- Inf**

**cor.est.ini <- tempo.cor**

**cor.est <- numeric(length = 5)**

**for(i6 in 1:5){** *# connected to cor.dec.per.loop*

**tempo.pos.est <- tempo.pos**

**pos <- sample.int(n = pos.selec.seq.max , size = count.est, replace = TRUE)** *# selection of n position*

**for(i7 in 1:count.est){**

**pos2 <- pos[i7]** *# selection of 1 position*

**tempo.pos.est[c(pos2 + 1, pos2)] <- tempo.pos.est[c(pos2, pos2 + 1)]**

**}**

**tempo.cor.est <- abs(cor(x = data1[tempo.pos.est], y = data2, use = "pairwise.complete.obs", method = cor.method))**

**cor.est[i6] <- tempo.cor.est**

**tempo.cor.dec.per.loop <- (cor.est.ini - tempo.cor.est) / count.est** *# correlation decrease per loop*

**if(is.na(tempo.cor.dec.per.loop) | ! is.finite(tempo.cor.dec.per.loop)){**

**tempo.cat <- paste0("ERROR IN ", function.name, ": CODE INCONSISTENCY 2\ncor.est.ini: ", cor.est.ini, "\ntempo.cor.est: ", tempo.cor.est)**

**stop(paste0("\n\n================\n\n", tempo.cat, "\n\n================\n\n", ifelse(is.null(warn), "", paste0("IN ADDITION\nWARNING", ifelse(warn.count > 1, "S", ""), ":\n\n", warn))), call. = FALSE)** *# == in stop() to be able to add several messages between ==*

**}**

**cor.dec.per.loop[i6] <- tempo.cor.dec.per.loop**

**}**

**cor.est <- cor.est[which.max(cor.dec.per.loop)]** *# max to avoid to go to far with for loop (tempo.cor below tempo.limit)*

**cor.dec.per.loop <- max(cor.dec.per.loop, na.rm = TRUE)** *# max to avoid to go to far with for loop (tempo.cor below tempo.limit)*

**loop.nb.est <- round((tempo.cor - cor.limit) / cor.dec.per.loop)**

**}else{**

**if(GOBACK == TRUE){**

**loop.nb.est <- round(loop.nb.est / 2)**

**}else{**

**cor.dec.per.loop <- (cor.ini - tempo.cor) / count**

**loop.nb.est <- round((tempo.cor - cor.limit) / cor.dec.per.loop)**

**}**

**}**

*# end estimation step*

*# loop step*

**if(is.na(loop.nb.est) | ! is.finite(loop.nb.est)){**

**tempo.cat <- paste0("ERROR IN ", function.name, ": CODE INCONSISTENCY 1\nloop.nb.est: ", loop.nb.est, "\ncor.ini: ", cor.ini, "\ntempo.cor: ", tempo.cor, "\ncor.limit: ", cor.limit, "\ncor.dec.per.loop: ", cor.dec.per.loop)**

**stop(paste0("\n\n================\n\n", tempo.cat, "\n\n================\n\n", ifelse(is.null(warn), "", paste0("IN ADDITION\nWARNING", ifelse(warn.count > 1, "S", ""), ":\n\n", warn))), call. = FALSE)** *# == in stop() to be able to add several messages between ==*

**}else if(loop.nb.est > 1e4){** *# below -> leave the while loop*

**tempo.pos.secu <- tempo.pos**

**count.secu <- count**

**tempo.cor.secu <- tempo.cor**

**cat(paste0("\n", ifelse(text.print == "", "", paste0(text.print, " | ")), "INITIAL SETTINGS BEFORE ROUND: ", round, " | LOOP COUNT: ", format(count, big.mark=","), " | GO BACK: ", GOBACK, " | LOOP NUMBER ESTIMATION: ", format(loop.nb.est, big.mark=","), " | CORRELATION LIMIT: ", fun\_round(cor.limit, 4), " | ABS TEMPO CORRELATION: ", fun\_round(tempo.cor, 4)))**

**print.count.loop <- logical(length = print.count)**

**print.count.loop[length(print.count.loop)] <- TRUE** *# not this to avoid long vector, but not forget to reset during printing: print.count.loop[(1:trunc(n / print.count) \* print.count)] <- TRUE # counter to speedup*

**count.loop <- 0**

**pos <- sample.int(n = pos.selec.seq.max , size = print.count, replace = TRUE)** *# selection of random positions. BEWARE: n = pos.selec.seq.max because already - 1 (see above) but is connected to tempo.pos[c(pos2 + 1, pos2)] <- tempo.pos[c(pos2, pos2 + 1)]*

**tempo.date.loop <- Sys.time()**

**tempo.time.loop <- as.numeric(tempo.date.loop)**

**for(i6 in 1:loop.nb.est){**

**count.loop <- count.loop + 1**

**pos2 <- pos[count.loop] # selection of 1 position**

**tempo.pos[c(pos2 + 1, pos2)] <- tempo.pos[c(pos2, pos2 + 1)]**

**if(print.count.loop[count.loop]){**

**count.loop <- 0**

**pos <- sample.int(n = pos.selec.seq.max , size = print.count, replace = TRUE)** *# BEWARE: never forget to resample here*

**tempo.time <- as.numeric(Sys.time())**

**tempo.lapse <- round(lubridate::seconds\_to\_period(tempo.time - tempo.time.loop))**

**final.loop <- (tempo.time - tempo.time.loop) / i6 \* loop.nb.est** *# expected duration in seconds # intra nb.compar loop lapse: time lapse / cycles done \* cycles remaining*

**final.exp <- as.POSIXct(final.loop, origin = tempo.date.loop)**

**cat(paste0("\n", ifelse(text.print == "", "", paste0(text.print, " | ")), "FOR LOOP | ROUND ", round, " | LOOP: ", format(i6, big.mark=","), " / ", format(loop.nb.est, big.mark=","), " | TIME SPENT: ", tempo.lapse, " | EXPECTED END: ", final.exp))**

**}**

**}**

**count <- count + loop.nb.est** *# out of the loop to speedup*

**tempo.cor <- abs(cor(x = data1[tempo.pos], y = data2, use = "pairwise.complete.obs", method = cor.method))**

**if(tempo.cor > tempo.cor.secu | ((tempo.cor - cor.limit) < 0 & abs(tempo.cor - cor.limit) > smallest.cor.dec \* round(log10(max(ini.pos, na.rm = TRUE))))){**

**GOBACK <- TRUE**

**tempo.pos <- tempo.pos.secu**

**count <- count.secu**

**tempo.cor <- tempo.cor.secu**

**}else{**

**GOBACK <- FALSE**

**}**

**}else{**

**cat(paste0("\n", ifelse(text.print == "", "", paste0(text.print, " | ")), "FINAL WHILE LOOP | LOOP COUNT: ", format(count, big.mark=","), " | CORRELATION LIMIT: ", fun\_round(cor.limit, 4), " | ABS TEMPO CORRELATION: ", fun\_round(tempo.cor, 4)))**

**print.count.loop <- logical(length = print.count)**

**print.count.loop[length(print.count.loop)] <- TRUE** *# counter to speedup*

**count.loop <- 0 #**

**pos <- sample.int(n = pos.selec.seq.max , size = print.count, replace = TRUE)** *# selection of random positions. BEWARE: n = pos.selec.seq.max because already - 1 (see above) but is connected to tempo.pos[c(pos2 + 1, pos2)] <- tempo.pos[c(pos2, pos2 + 1)]*

**tempo.cor.loop <- tempo.cor**

**tempo.date.loop <- Sys.time()**

**tempo.time.loop <- as.numeric(tempo.date.loop)**

**while(tempo.cor > cor.limit){**

**count <- count + 1**

**count.loop <- count.loop + 1**

**pos2 <- pos[count.loop]**

**tempo.pos[c(pos2 + 1, pos2)] <- tempo.pos[c(pos2, pos2 + 1)]**

**tempo.cor <- abs(cor(x = data1[tempo.pos], y = data2, use = "pairwise.complete.obs", method = cor.method))**

**if(print.count.loop[count.loop]){**

**count.loop <- 0**

**pos <- sample.int(n = pos.selec.seq.max , size = print.count, replace = TRUE)** *# BEWARE: never forget to resample here*

**tempo.time <- as.numeric(Sys.time())**

**tempo.lapse <- round(lubridate::seconds\_to\_period(tempo.time - tempo.time.loop))**

**final.loop <- (tempo.time - tempo.time.loop) / (tempo.cor.loop - tempo.cor) \* (tempo.cor - cor.limit)** *# expected duration in seconds # tempo.cor.loop - tempo.cor always positive and tempo.cor decreases progressively starting from tempo.cor.loop*

**final.exp <- as.POSIXct(final.loop, origin = tempo.date.loop)**

**cat(paste0("\n", ifelse(text.print == "", "", paste0(text.print, " | ")), "WHILE LOOP | LOOP NB: ", format(count.loop, big.mark=","), " | COUNT: ", format(count, big.mark=","), " | CORRELATION LIMIT: ", fun\_round(cor.limit, 4), " | ABS TEMPO CORRELATION: ", fun\_round(tempo.cor, 4), " | TIME SPENT: ", tempo.lapse, " | EXPECTED END: ", final.exp))**

**}**

**}**

**}**

**}**

**tempo.time <- as.numeric(Sys.time())**

**tempo.lapse <- round(lubridate::seconds\_to\_period(tempo.time - ini.time))**

**cat(paste0("\n", ifelse(text.print == "", "", paste0(text.print, " | ")), "WHILE/FOR LOOPS END | LOOP COUNT: ", format(count, big.mark=","), " | NB OF ROUNDS: ", round, " | CORRELATION LIMIT: ", fun\_round(cor.limit, 4), " | ABS TEMPO CORRELATION: ", fun\_round(tempo.cor, 4), " | TOTAL SPENT TIME: ", tempo.lapse))**

**}**

**tempo.cor <- ifelse(neg.cor == TRUE, -tempo.cor, tempo.cor)**

**}**

**}**

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

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

**warning(paste0("FROM ", function.name, ":\n\n", warn), call. = FALSE)**

**}**

**output <- list(data = data1[tempo.pos], warn = warn, cor = if(is.null(data2)){cor(ini.pos, tempo.pos, method = "spearman")}else{tempo.cor}, count = count)**

**return(output)**

**}**

######## fun\_slide() #### return a computation made on a vector using a sliding window

**fun\_slide <- function(data, window.size, step, from = NULL, to = NULL, fun, args = NULL, boundary = "left", parall = FALSE, thread.nb = NULL, print.count = 100, res.path = NULL, lib.path = NULL, verbose = TRUE, cute.path = "C:\\Users\\Gael\\Documents\\Git\_projects\\cute\_little\_R\_functions\\cute\_little\_R\_functions.R"){**

*# AIM*

*# return a computation made on a vector using a sliding window*

*# WARNING*

*# The function uses two strategies, depending on the amout of memory required which depends on the data, window.size and step arguments. The first one uses lapply(), is generally fast but requires lots of memory. The second one uses a parallelized loop. The choice between the two strategies is automatic if parall argument is FALSE, and is forced toward parallelization if parall argument is TRUE*

*# The parall argument forces the parallelization, which is convenient when the data argument is big, because the lapply() function is sometimes slower than the parallelization*

*# ARGUMENTS*

*# data: vector, matrix, table or array of numeric values (mode must be numeric). Inf not allowed. NA will be removed before computation*

*# window.size: single numeric value indicating the width of the window sliding across data (in the same unit as data value)*

*# step: single numeric value indicating the step between each window (in the same unit as data value). Cannot be larger than window.size*

*# from: value of the left boundary of the first sliding window. If NULL, min(data) is used. The first window will strictly have from or min(data) as left boundary*

*# to: value of the right boundary of the last sliding window. If NULL, max(data) is used. Warning: (1) the final last window will not necessary have to|max(data) as right boundary. In fact the last window will be the one that contains to|max(data) for the first time, i.e., min[from|min(data) + window.size + n \* step >= to|max(data)]; (2) In fact, the >= in min[from|min(data) + window.size + n \* step >= to|max(data)] depends on the boundary argument (>= for "right" and > for "left"); (3) to have the rule (1) but for the center of the last window, use to argument as to = to|max(data) + window.size / 2*

*# fun: function or character string (without brackets) indicating the name of the function to apply in each window. Example: fun = "mean", or fun = mean*

*# args: character string of additional arguments of fun (separated by a comma between the quotes). Example args = "na.rm = TRUE" for fun = mean. Ignored if NULL*

*# boundary: either "left" or "right". Indicates if the sliding window includes values equal to left boundary and exclude values equal to right boundary ("left") or the opposite ("right")*

*# parall: logical. Force parallelization ?*

*# thread.nb: numeric value indicating the number of threads to use if ever parallelization is required. If NULL, all the available threads will be used*

*# print.count: interger value. Print a working progress message every print.count during loops. BEWARE: can increase substentially the time to complete the process using a small value, like 10 for instance. Use Inf is no loop message desired*

*# res.path: character string indicating the absolute pathway where the parallelization log file will be created if parallelization is used. If NULL, will be created in the R current directory*

*# lib.path: character vector specifying the absolute pathways of the directories containing the required packages if not in the default directories. Ignored if NULL*

*# verbose: logical. Display messages?*

*# cute.path: character string indicating the absolute path of the cute.R file. Will be remove when cute will be a package. Not considered if thread.nb is NULL*

*# REQUIRED PACKAGES*

*# lubridate*

*# parallel if parallelization is used*

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

*# fun\_check()*

*# fun\_get\_message*

*# fun\_pack()*

*# RETURN*

*# a data frame containing*

*#$left : the left boundary of each window (in the unit of the data argument)*

*#$right : the right boundary of each window (in the unit of data argument)*

*#$center : the center of each window (in the unit of data argument)*

*#$value : the computed value by the fun argument in each window)*

*# EXAMPLES*

*# fun\_slide(data = c(1:10, 100:110, 500), window.size = 5, step = 2, fun = length, boundary = "left")*

*# fun\_slide(data = c(1:10, 100:110, 500), window.size = 5, step = 2, fun = length, boundary = "right") # effect of boundary argument*

*# fun\_slide(data = c(1:10, 100:110, 500), window.size = 5, step = 2, fun = length, boundary = "left", parall = TRUE) # effect of parall argument*

*# DEBUGGING*

*# data = c(1:10, 100:110, 500) ; window.size = 5 ; step = 2 ; from = NULL ; to = NULL ; fun = length ; args = NULL ; boundary = "left" ; parall = FALSE ; thread.nb = NULL ; print.count = 100 ; res.path = NULL ; lib.path = NULL ; verbose = TRUE ; cute.path = "C:\\Users\\Gael\\Documents\\Git\_projects\\cute\_little\_R\_functions\\cute\_little\_R\_functions.R"*

*# data = lag.pos; window.size = window.size; step = step; fun = length; from = min(a$pos); to = max(a$pos)*

*# function name*

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

**instruction <- match.call()**

*# end function name*

*# required function checking*

**req.function <- c(**

**"fun\_check",**

**"fun\_get\_message",**

**"fun\_pack"**

**)**

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

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

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

**stop(paste0("\n\n================\n\n", tempo.cat, "\n\n================\n\n"), call. = FALSE)** *# == in stop() to be able to add several messages between ==*

**}**

**}**

*# end required function checking*

*# argument primary checking*

*# arg with no default values*

**if(any(missing(data) | missing(window.size) | missing(step) | missing(fun))){**

**tempo.cat <- paste0("ERROR IN ", function.name, ": ARGUMENTS fun, args AND val HAVE NO DEFAULT VALUE AND REQUIRE ONE")**

**stop(paste0("\n\n================\n\n", tempo.cat, "\n\n================\n\n"), call. = FALSE)** *# == in stop() to be able to add several messages between ==*

**}**

*# end arg with no default values*

*# using fun\_check()*

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

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

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

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

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

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

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

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

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

**}**

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

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

**}**

**tempo1 <- fun\_check(data = fun, class = "vector", mode = "character", length = 1, fun.name = function.name)**

**tempo2 <- fun\_check(data = fun, class = "function", length = 1, fun.name = function.name)**

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

**tempo.cat <- paste0("ERROR IN ", function.name, ": fun ARGUMENT MUST BE A FUNCTION OR A CHARACTER STRING OF THE NAME OF A FUNCTION")**

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

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

**}**

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

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

**}**

**tempo <- fun\_check(data = boundary, options = c("left", "right"), length = 1, fun.name = function.name) ; eval(ee)**

**if( ! is.null(thread.nb)){**

**tempo <- fun\_check(data = thread.nb, typeof = "integer", double.as.integer.allowed = TRUE, neg.values = FALSE, length = 1, fun.name = function.name) ; eval(ee)**

**}**

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

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

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

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

**}**

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

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

**}**

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

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

**if( ! file.exists(cute.path)){**

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

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

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

**}**

**}**

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

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

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

**}**

*# end using fun\_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\_check\_dev)) # activate this line and use the function (with no arguments left as NULL) to check arguments status and if they have been checked using fun\_check()*

*# end argument primary checking*

*# second round of checking and data preparation*

*# dealing with NA*

**if(any(is.na(window.size)) | any(is.na(step)) | any(is.na(from)) | any(is.na(to)) | suppressWarnings(any(is.na(fun))) | any(is.na(args)) | any(is.na(boundary)) | any(is.na(parall)) | any(is.na(thread.nb)) | any(is.na(print.count)) | any(is.na(res.path)) | any(is.na(lib.path)) | any(is.na(verbose))){**

**tempo.cat <- paste0("ERROR IN ", function.name, ": NO ARGUMENT EXCEPT data CAN HAVE NA VALUES\nPROBLEMATIC ARGUMENTS ARE: ", paste(c("window.size", "step", "from", "to", "fun", "args", "boundary", "parall", "thread.nb", "print.count", "res.path", "lib.path", "verbose")[c(any(is.na(window.size)), any(is.na(step)), any(is.na(from)), any(is.na(to)), suppressWarnings(any(is.na(fun))), any(is.na(args)), any(is.na(boundary)), any(is.na(parall)), any(is.na(thread.nb)), any(is.na(print.count)), any(is.na(res.path)), any(is.na(lib.path)), any(is.na(verbose)))], collapse = "\n"))**

**stop(paste0("\n\n================\n\n", tempo.cat, "\n\n================\n\n"), call. = FALSE)** *# == in stop() to be able to add several messages between ==*

**}**

*# end dealing with NA*

*# dealing with NULL*

**if(is.null(data) | is.null(window.size) | is.null(step) | is.null(fun) | is.null(boundary) | is.null(parall) | is.null(print.count) | is.null(verbose)){**

**tempo.cat <- paste0("ERROR IN ", function.name, ": THESE ARGUMENTS\ndata\nwindow.size\nstep\nfun\nboundary\nparall\nprint.count\nverbose\nCANNOT BE NULL\nPROBLEMATIC ARGUMENTS ARE: ", paste(c("data", "window.size", "step", "fun", "boundary", "parall", "print.count", "verbose")[c(is.null(data), is.null(window.size), is.null(step), is.null(fun), is.null(boundary), is.null(parall), is.null(print.count), is.null(verbose))], collapse = "\n"))**

**stop(paste0("\n\n================\n\n", tempo.cat, "\n\n================\n\n"), call. = FALSE)** *# == in stop() to be able to add several messages between ==*

**}**

*# end dealing with NULL*

**if(any( ! is.finite(data))){**

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

**stop(paste0("\n\n================\n\n", tempo.cat, "\n\n================\n\n"), call. = FALSE)**

**}**

**if(step > window.size){**

**tempo.cat <- paste0("ERROR IN ", function.name, ": step ARGUMENT MUST BE LOWER THAN window.size ARGUMENT\nstep: ", paste(step, collapse = " "), "\nwindow.size: ", paste(window.size, collapse = " "))**

**stop(paste0("\n\n================\n\n", tempo.cat, "\n\n================\n\n"), call. = FALSE)**

**}**

**if( ! is.null(thread.nb)){**

**if(thread.nb < 1){**

**tempo.cat <- paste0("ERROR IN ", function.name, ": thread.nb PARAMETER MUST EQUAL OR GREATER THAN 1: ", thread.nb)**

**stop(paste0("\n\n================\n\n", tempo.cat, "\n\n================\n\n"), call. = FALSE)**

**}**

**}**

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

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

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

**stop(paste0("\n\n================\n\n", tempo.cat, "\n\n================\n\n"), call. = FALSE)**

**}**

**}else{**

**res.path <- getwd() # working directory**

**}**

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

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

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

**stop(paste0("\n\n================\n\n", tempo.cat, "\n\n================\n\n"), call. = FALSE)**

**}**

**}**

*# end second round of checking and data preparation*

*# package checking*

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

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

*# end package checking*

*# main code*

**if(verbose == TRUE){**

**cat("\nfun\_slide JOB IGNITION\n")**

**}**

**ini.date <- Sys.time()**

**ini.time <- as.numeric(ini.date)** *# time of process begin, converted into seconds*

**fun <- match.fun(fun)** *# make fun <- get(fun) is fun is a function name written as character string of length 1*

**if(boundary == "left"){**

**left <- ">="**

**right <- "<"**

**right.last.wind <- ">"**

**}else if(boundary == "right"){**

**left <- ">"**

**right <- "<="**

**right.last.wind <- ">="**

**}else{**

**tempo.cat <- paste0("INTERNAL CODE ERROR IN ", function.name, "\nCODE INCONSISTENCY 1")**

**stop(paste0("\n\n================\n\n", tempo.cat, "\n\n================\n\n"), call. = FALSE)** *# == in stop() to be able to add several messages between ==*

**}**

**data <- as.vector(data)**

**data <- sort(data, na.last = NA)** *# NA removed*

**wind <- data.frame(left = seq(from = if(is.null(from)){min(data, na.rm = TRUE)}else{from}, to = if(is.null(to)){max(data, na.rm = TRUE)}else{to}, by = step))**

**wind <- data.frame(wind, right = wind$left + window.size)**

**wind <- data.frame(wind, center = (wind$left + wind$right) / 2)**

**if(all(wind$right < if(is.null(to)){max(data, na.rm = TRUE)}else{to})){**

**tempo.cat <- paste0("INTERNAL CODE ERROR IN ", function.name, "\nCODE INCONSISTENCY 2")**

**stop(paste0("\n\n================\n\n", tempo.cat, "\n\n================\n\n"), call. = FALSE)** *# == in stop() to be able to add several messages between ==*

**}**

*# The 3 next lines is for the rule of to argument with center (see to argument description)*

*# if(any(wind$center > max(data, na.rm = TRUE))){*

*# wind <- wind[ ! wind$center > max(data, na.rm = TRUE),]*

*# }*

**if(sum(get(right.last.wind)(wind$right, if(is.null(to)){max(data, na.rm = TRUE)}else{to}), na.rm = TRUE) > 1){**

**tempo.log <- get(right.last.wind)(wind$right, if(is.null(to)){max(data, na.rm = TRUE)}else{to})**

**tempo.log[min(which(tempo.log), na.rm = TRUE)] <- FALSE** *# convert the first left boundary that goes above max(data, na.rm = TRUE) to FALSE to keep it (the next ones will be removed)*

**wind <- wind[ ! tempo.log,]**

**}**

*# test if lapply can be used*

**if(parall == FALSE){**

*# new environment*

**env.name <- paste0("env", ini.time)**

**if(exists(env.name, where = -1)){**

**tempo.cat <- paste0("ERROR IN ", function.name, ": ENVIRONMENT env.name ALREADY EXISTS. PLEASE RERUN ONCE")**

**stop(paste0("\n\n================\n\n", tempo.cat, "\n\n================\n\n"), call. = FALSE)** *# == in stop() to be able to add several messages between ==*

**}else{**

**assign(env.name, new.env())**

**assign("wind", wind, envir = get(env.name))**

**assign("data", data, envir = get(env.name))**

**}**

*# end new environment*

**tempo.message <- fun\_get\_message(data="lapply(X = wind$left, Y = data, FUN = function(X, Y){res <- get(left)(Y, X) ; return(res)})", kind = "error", header = FALSE, env = get(env.name), print.no = FALSE)**

**rm(env.name)** *# optional, because should disappear at the end of the function execution*

**}else{**

**tempo.message <- "ERROR"** *# with this, force the parallelization by default*

**}**

*# end test if lapply can be used*

**if( ! any(grepl(x = tempo.message, pattern = "ERROR.\*"))){**

**left.log <- lapply(X = wind$left, Y = data, FUN = function(X, Y){**

**res <- get(left)(Y, X)**

**return(res)**

**})**

**right.log <- lapply(X = wind$right, Y = data, FUN = function(X, Y){**

**res <- get(right)(Y, X)**

**return(res)**

**})**

**log <- mapply(FUN = "&", left.log, right.log, SIMPLIFY = FALSE)**

**output <- eval(parse(text = paste0("sapply(lapply(log, FUN = function(X){(data[X])}), FUN = fun", if( ! is.null(args)){paste0(", ", args)}, ")")))** *# take the values of the data vector according to log (list of logical, each compartment of length(data)) and apply fun with args of fun*

**if(length(output) != nrow(wind)){**

**tempo.cat <- paste0("INTERNAL CODE ERROR IN ", function.name, "\nCODE INCONSISTENCY 3")**

**stop(paste0("\n\n================\n\n", tempo.cat, "\n\n================\n\n"), call. = FALSE)** *# == in stop() to be able to add several messages between ==*

**}else{**

**output <- data.frame(wind, value = output)**

**}**

**}else{**

**if(verbose == TRUE){**

**tempo.cat <- paste0("PARALLELIZATION INITIATED AT: ", ini.date)**

**cat(paste0("\n", tempo.cat, "\n"))**

**}**

**tempo.thread.nb = parallel::detectCores(all.tests = FALSE, logical = TRUE)** *# detect the number of threads*

**if( ! is.null(thread.nb)){**

**if(tempo.thread.nb < thread.nb){**

**thread.nb <- tempo.thread.nb**

**if(verbose == TRUE){**

**tempo.cat <- paste0("ONLY: ", tempo.thread.nb, " THREADS AVAILABLE")**

**cat(paste0("\n", tempo.cat, "\n"))**

**}**

**}**

**}else{**

**thread.nb <- tempo.thread.nb**

**}**

**if(verbose == TRUE){**

**tempo.cat <- paste0("NUMBER OF THREADS USED: ", thread.nb)**

**cat(paste0("\n ", tempo.cat, "\n"))**

**}**

**Clust <- parallel::makeCluster(thread.nb, outfile = paste0(res.path, "/fun\_slide\_parall\_log.txt"))** *# outfile to print or cat during parallelization (only possible in a file, outfile = "" do not work on windows)*

**cluster.list <- parallel::clusterSplit(Clust, 1:nrow(wind))** *# split according to the number of cluster*

**if(verbose == TRUE){**

**tempo.cat <- paste0("SPLIT OF TEST NUMBERS IN PARALLELISATION:")**

**cat(paste0("\n ", tempo.cat, "\n"))**

**str(cluster.list)** *# using print(str()) add a NULL below the result*

**cat("\n")**

**}**

**paral.output.list <- parallel::clusterApply(** *#*

**cl = Clust,**

**x = cluster.list,**

**function.name = function.name,**

**data = data,**

**FUN = fun, # because fun argument of clusterApply**

**args = args,**

**thread.nb = thread.nb,**

**print.count = print.count,**

**wind = wind,**

**left = left,**

**right = right,**

**res.path = res.path,**

**lib.path = lib.path,**

**verbose = verbose,**

**cute.path = cute.path,**

**fun = function(**

**x,**

**function.name,**

**data,**

**FUN,**

**args,**

**thread.nb,**

**print.count,**

**wind,**

**left,**

**right,**

**res.path,**

**lib.path,**

**verbose,**

**cute.path**

**){**

*# check again: very important because another R*

**process.id <- Sys.getpid()**

**cat(paste0("\nPROCESS ID ", process.id, " -> TESTS ", x[1], " TO ", x[length(x)], "\n"))**

**source(cute.path, local = .GlobalEnv)**

**fun\_pack(req.package = "lubridate", lib.path = lib.path, load = TRUE)** *# load = TRUE to be sure that functions are present in the environment. And this prevent to use R.lib.path argument of fun\_python\_pack()*

*# end check again: very important because another R*

**ini.date <- Sys.time()**

**ini.time <- as.numeric(ini.date)** *# time of process begin, converted into*

**output <- NULL**

**print.count.loop <- 0**

**for(i4 in 1:length(x)){**

**print.count.loop <- print.count.loop + 1**

**log <- get(left)(data, wind$left[x[i4]]) & get(right)(data, wind$right[x[i4]])**

**output <- c(output, eval(parse(text = paste0("FUN(data[log]", if( ! is.null(args)){paste0(", ", args)}, ")"))))**

**if(verbose == TRUE){**

**if(print.count.loop == print.count){**

**print.count.loop <- 0**

**tempo.time <- as.numeric(Sys.time())**

**tempo.lapse <- round(lubridate::seconds\_to\_period(tempo.time - ini.time))**

**final.loop <- (tempo.time - ini.time) / i4 \* length(x)** *# expected duration in seconds # intra nb.compar loop lapse: time lapse / cycles done \* cycles remaining*

**final.exp <- as.POSIXct(final.loop, origin = ini.date)**

**cat(paste0("\nIN PROCESS ", process.id, " | LOOP ", format(i4, big.mark=","), " / ", format(length(x), big.mark=","), " | TIME SPENT: ", tempo.lapse, " | EXPECTED END: ", final.exp))**

**}**

**if(i4 == length(x)){**

**tempo.time <- as.numeric(Sys.time())**

**tempo.lapse <- round(lubridate::seconds\_to\_period(tempo.time - ini.time))**

**cat(paste0("\nPROCESS ", process.id, " ENDED | LOOP ", format(i4, big.mark=","), " / ", format(length(x), big.mark=","), " | TIME SPENT: ", tempo.lapse, "\n\n"))**

**}**

**}**

**}**

**wind <- wind[x, ]**

**if(length(output) != nrow(wind)){**

**tempo.cat <- paste0("INTERNAL CODE ERROR IN ", function.name, "\nCODE INCONSISTENCY 4")**

**stop(paste0("\n\n================\n\n", tempo.cat, "\n\n================\n\n"), call. = FALSE)** *# == in stop() to be able to add several messages between ==*

**}else{**

**output <- data.frame(wind, value = output)**

**return(output)**

**}**

**}**

**)**

**parallel::stopCluster(Clust)**

*# result assembly*

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

**for(i2 in 1:length(paral.output.list)){** *# compartment relatives to each parallelization*

**output <- rbind(output, paral.output.list[[i2]])**

**}**

*# end result assembly*

**if(nrow(output) != nrow(wind)){**

**tempo.cat <- paste0("INTERNAL CODE ERROR IN ", function.name, "\nCODE INCONSISTENCY 5\nlength(output): ", length(output), "\nnrow(wind): ", nrow(wind))**

**stop(paste0("\n\n================\n\n", tempo.cat, "\n\n================\n\n"), call. = FALSE)** *# == in stop() to be able to add several messages between ==*

**}else{**

**output <- output[order(output$left), ]**

**}**

**}**

**if(verbose == TRUE){**

**end.date <- Sys.time()**

**end.time <- as.numeric(end.date)**

**total.lapse <- round(lubridate::seconds\_to\_period(end.time - ini.time))**

**cat(paste0("fun\_slide JOB END\n\nTIME: ", end.date, "\n\nTOTAL TIME LAPSE: ", total.lapse, "\n\n\n"))**

**}**

**return(output)**

**}**

################ Graphics management

*# this order can be used:*

*# fun\_width()*

*# fun\_open()*

*# 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()*

######## fun\_width() #### window width depending on classes to plot

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

**fun\_width <- 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\_width()*

*# fun\_open()*

*# 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()*

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

*# fun\_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\_width(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(utils::find("fun\_check", mode = "function")) == 0){**

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

**stop(paste0("\n\n================\n\n", tempo.cat, "\n\n================\n\n"), call. = FALSE)** *# == in stop() to be able to add several messages between ==*

**}**

*# end required function checking*

*# argument checking*

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

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

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

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

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

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

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

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

**tempo <- fun\_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(paste0("\n\n================\n\n", paste(text.check[arg.check], collapse = "\n"), "\n\n================\n\n"), call. = FALSE)** *#*

**}**

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

*# end argument checking*

*# 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() #### open a GUI or pdf graphic window

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

**fun\_open <- function(pdf = TRUE, pdf.path = "working.dir", pdf.name = "graph", width = 7, height = 7, paper = "special", pdf.overwrite = FALSE, rescale = "fixed", remove.read.only = TRUE, return.output = FALSE){**

*# AIM*

*# open a pdf or screen (GUI) graphic window and return initial graphic parameters*

*# this order can be used:*

*# fun\_width()*

*# fun\_open()*

*# 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()*

*#WARNING*

*# On Linux, use pdf = 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*

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

*# fun\_check()*

*# ARGUMENTS:*

*# pdf: logical. Use pdf display? If FALSE, a GUI is opened*

*# pdf.path: where the pdf is saved (do not terminate by / or \\). Write "working.dir" if working directory is required (default). Ignored if pdf == FALSE*

*# pdf.name: name of the pdf file containing the graphs (the .pdf extension is added by the function, if not detected in the name end). Ignored if pdf == FALSE*

*# width: width of the window (in inches)*

*# height: height of the window (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 the paper dimension will be width and height. With another paper format, if width or height is over the size of the paper, width or height will be modified such that the plot is adjusted to the paper dimension (see $dim in the returned list below to see the modified dimensions). Ignored if pdf == FALSE*

*# pdf.overwrite: logical. Existing pdf can be overwritten? . Ignored if pdf == FALSE*

*# rescale: kind of GUI. Either "R", "fit", or "fixed". Ignored on Mac and Linux OS. See ?windows for details*

*# remove.read.only: logical. remove the read only (R.O.) graphical parameters? If TRUE, the graphical parameters are returned without the R.O. parameters. The returned $ini.par list can be used to set the par() of a new graphical device. If FALSE, graphical parameters are returned with the R.O. parameters, which provides information like text dimension (see ?par() ). The returned $ini.par list can be used to set the par() of a new graphical device, but generate a warning message. Ignored if return.output == FALSE.*

*# return.output: logical. Return output ? If TRUE the output list is displayed*

*# RETURN*

*# a list containing:*

*# $pdf.loc: path of the pdf created*

*# $ini.par: initial par() parameters*

*# $zone.ini: initial window spliting*

*# $dim: dimension of the graphical device (in inches)*

*# EXAMPLES*

*# fun\_open(pdf = FALSE, pdf.path = "C:/Users/Gael/Desktop", pdf.name = "graph", width = 7, height = 7, paper = "special", pdf.overwrite = FALSE, return.output = TRUE)*

*# DEBUGGING*

*# pdf = TRUE ; pdf.path = "C:/Users/Gael/Desktop" ; pdf.name = "graphs" ; width = 7 ; height = 7 ; paper = "special" ; pdf.overwrite = FALSE ; remove.read.only = 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(utils::find("fun\_check", mode = "function")) == 0){**

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

**stop(paste0("\n\n================\n\n", tempo.cat, "\n\n================\n\n"), call. = FALSE)** *# == in stop() to be able to add several messages between ==*

**}**

*# end required function checking*

*# argument checking*

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

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

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

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

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

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

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

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

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

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

**tempo <- fun\_check(data = rescale, options = c("R", "fit", "fixed"), length = 1, fun.name = function.name) ; eval(ee)**

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

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

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

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

**}**

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

*# end argument checking*

*# main code*

**if(pdf.path == "working.dir"){**

**pdf.path <- getwd()**

**}else{**

**if(grepl(x = pdf.path, pattern = ".+/$")){**

**pdf.path <- sub(x = pdf.path, pattern = "/$", replacement = "")** *# remove the last /*

**}else if(grepl(x = pdf.path, pattern = ".+[\\]$")){** *# or ".+\\\\$" # cannot be ".+\$" because \$ does not exist contrary to \n*

**pdf.path <- sub(x = pdf.path, pattern = "[\\]$", replacement = "")** *# remove the last /*

**}**

**if(dir.exists(pdf.path) == FALSE){**

**tempo.cat <- paste0("ERROR IN ", function.name, "\npdf.path ARGUMENT DOES NOT CORRESPOND TO EXISTING DIRECTORY\n", pdf.path)**

**stop(paste0("\n\n================\n\n", tempo.cat, "\n\n================\n\n"), call. = FALSE)** *# == in stop() to be able to add several messages between ==*

**}**

**}**

*# par.ini recovery*

*# cannot use pdf(file = NULL), because some small differences between pdf() and other devices. For instance, differences with windows() for par()$fin, par()$pin and par()$plt*

**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 = remove.read.only)** *# 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 == TRUE){**

**tempo.code <- 0**

**while(file.exists(paste0(pdf.path, "/recover\_ini\_par", tempo.code, ".pdf")) == TRUE){**

**tempo.code <- tempo.code + 1**

**}**

**pdf(width = width, height = height, file=paste0(pdf.path, "/recover\_ini\_par", tempo.code, ".pdf"), paper = paper)**

**ini.par <- par(no.readonly = remove.read.only)** *# 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 window*

**file.remove(paste0(pdf.path, "/recover\_ini\_par", tempo.code, ".pdf"))** *# remove the pdf file*

**}else{**

*# test if X11 can be opened*

**if(file.exists(paste0(getwd(), "/Rplots.pdf"))){**

**tempo.cat <- paste0("ERROR IN ", function.name, "\nTHIS FUNCTION CANNOT BE USED ON LINUX IF A Rplots.pdf FILE ALREADY EXISTS HERE\n", getwd())**

**stop(paste0("\n\n================\n\n", tempo.cat, "\n\n================\n\n"), call. = FALSE)** *# == in stop() to be able to add several messages between ==*

**}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 = remove.read.only)** *# 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 <- ("ERROR IN fun\_open()\nTHIS FUNCTION CANNOT OPEN GUI ON LINUX OR NON MACOS UNIX SYSTEM (X GRAPHIC INTERFACE HAS TO BE SET)\nTO OVERCOME THIS, PLEASE SET pdf ARGUMENT TO TRUE AND RERUN")**

**stop(paste0("\n\n================\n\n", tempo.cat, "\n\n================\n\n"), call. = FALSE)** *# == in stop() to be able to add several messages between ==*

**}**

**}**

**}**

**}else{**

**open.fail <- NULL**

**quartz()**

**ini.par <- par(no.readonly = remove.read.only)** *# 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*

**}**

*# end par.ini recovery*

**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 == TRUE){**

**if(grepl(x = pdf.name, pattern = "\\.pdf$")){**

**pdf.name <- sub(x = pdf.name, pattern = "\\.pdf$", replacement = "")** *# remove the last .pdf*

**}**

**pdf.loc <- paste0(pdf.path, "/", pdf.name, ".pdf")**

**if(file.exists(pdf.loc) == TRUE & pdf.overwrite == FALSE){**

**tempo.cat <- paste0("ERROR IN ", function.name, "\npdf.loc FILE ALREADY EXISTS AND CANNOT BE OVERWRITTEN DUE TO pdf.overwrite ARGUMENT SET TO TRUE\n", pdf.loc)**

**stop(paste0("\n\n================\n\n", tempo.cat, "\n\n================\n\n"), call. = FALSE)** *# == in stop() to be able to add several messages between ==*

**}else{**

**pdf(width = width, height = height, file=pdf.loc, paper = paper)**

**}**

**}else if(pdf == 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, height = height, rescale = rescale)**

**}else if(Sys.info()["sysname"] == "Linux"){**

**if( ! is.null(open.fail)){**

**tempo.cat <- "ERROR IN fun\_open()\nTHIS FUNCTION CANNOT OPEN GUI ON LINUX OR NON MACOS UNIX SYSTEM (X GRAPHIC INTERFACE HAS TO BE SET)\nTO OVERCOME THIS, PLEASE SET pdf ARGUMENT TO TRUE AND RERUN"**

**stop(paste0("\n\n================\n\n", tempo.cat, "\n\n================\n\n"), call. = FALSE)** *# == in stop() to be able to add several messages between ==*

**}else{**

**X11(width = width, height = height)**

**}**

**}else{**

**quartz(width = width, height = height)**

**}**

**}**

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

**output <- list(pdf.loc = pdf.loc, ini.par = ini.par, zone.ini = zone.ini, dim = dev.size())**

**return(output)**

**}**

**}**

######## fun\_prior\_plot() #### set graph param before plotting (erase axes for instance)

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

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

**stop(paste0("\n\n================\n\n", tempo.cat, "\n\n================\n\n"), call. = FALSE)** *# == in stop() to be able to add several messages between ==*

**}**

*# end required function checking*

*# argument checking*

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

**tempo <- fun\_check(data = box.type, options = c("o", "l", "7", "c", "u", "]", "n"), length = 1, fun.name = function.name) ; eval(ee)**

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

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

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

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

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

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

**}**

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

*# end argument checking*

*# main code*

**if(is.null(dev.list())){**

**tempo.cat <- paste0("ERROR IN ", function.name, ": THIS FUNCTION CANNOT BE USED IF NO GRAPHIC DEVICE ALREADY OPENED (dev.list() IS CURRENTLY NULL)")**

**stop(paste0("\n\n================\n\n", tempo.cat, "\n\n================\n\n"), call. = FALSE)** *# == in stop() to be able to add several messages between ==*

**}**

*# par.ini recovery*

*# cannot use pdf(file = NULL), because some small differences between pdf() and other devices. For instance, differences with windows() for par()$fin, par()$pin and par()$plt*

**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 = FALSE)** *# 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("ERROR 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())**

**stop(paste0("\n\n================\n\n", tempo.cat, "\n\n================\n\n"), call. = FALSE)** *# == in stop() to be able to add several messages between ==*

**}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 = FALSE)** *# 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 = FALSE)** *# 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 <- ("ERROR IN fun\_prior\_plot()\nTHIS 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")**

**stop(paste0("\n\n================\n\n", tempo.cat, "\n\n================\n\n"), call. = FALSE)** *# == in stop() to be able to add several messages between ==*

**}**

**}**

**}else{** *# macOS*

**quartz()**

**ini.par <- par(no.readonly = FALSE)** *# 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")){**

**invisible(dev.set(active.wind.nb))** *# go back to the active window if exists*

**par(ini.par)** *# apply the initial par to current window*

**}**

**}**

*# end par.ini recovery*

**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\_scale() #### select nice label numbers when setting number of ticks on an axis



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

**fun\_scale <- function(n, lim, kind = "approx", lib.path = NULL){**

*# AIM*

*# attempt to select nice scale numbers when setting n ticks on a lim axis range*

*# ARGUMENTS*

*# n: desired number of main ticks on the axis (integer above 0)*

*# lim: vector of 2 numbers indicating the limit range of the axis. Order of the 2 values matters (for inverted axis). Can be log transformed values*

*# kind: either "approx" (approximative), "strict" (strict) or "strict.cl" (strict clean). If "approx", use the scales::trans\_breaks() function to provide an easy to read scale of approximately n ticks spanning the range of the lim argument. If "strict", cut the range of the lim argument into n + 1 equidistant part and return the n numbers at each boundary. This often generates numbers uneasy to read. If "strict.cl", provide an easy to read scale of exactly n ticks, but sometimes not completely spanning the range of the lim argument*

*# lib.path: character vector specifying the absolute pathways of the directories containing the required packages if not in the default directories. Ignored if NULL*

*# REQUIRED PACKAGES*

*# if kind = "approx":*

*# ggplot2*

*# scales*

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

*# fun\_check()*

*# fun\_round()*

*# RETURN*

*# a vector of numbers*

*# EXAMPLES*

*# approximate number of main ticks*

*# ymin = 2 ; ymax = 3.101 ; n = 5 ; scale <- fun\_scale(n = n, lim = c(ymin, ymax), kind = "approx") ; scale ; par(yaxt = "n", yaxs = "i", las = 1) ; plot(ymin:ymax, ymin:ymax, xlim = range(scale, ymin, ymax)[order(c(ymin, ymax))], ylim = range(scale, ymin, ymax)[order(c(ymin, ymax))], xlab = "DEFAULT SCALE", ylab = "NEW SCALE") ; par(yaxt = "s") ; axis(side = 2, at = scale)*

*# strict number of main ticks*

*# ymin = 2 ; ymax = 3.101 ; n = 5 ; scale <- fun\_scale(n = n, lim = c(ymin, ymax), kind = "strict") ; scale ; par(yaxt = "n", yaxs = "i", las = 1) ; plot(ymin:ymax, ymin:ymax, xlim = range(scale, ymin, ymax)[order(c(ymin, ymax))], ylim = range(scale, ymin, ymax)[order(c(ymin, ymax))], xlab = "DEFAULT SCALE", ylab = "NEW SCALE") ; par(yaxt = "s") ; axis(side = 2, at = scale)*

*# strict "clean" number of main ticks*

*# ymin = 2 ; ymax = 3.101 ; n = 5 ; scale <- fun\_scale(n = n, lim = c(ymin, ymax), kind = "strict.cl") ; scale ; par(yaxt = "n", yaxs = "i", las = 1) ; plot(ymin:ymax, ymin:ymax, xlim = range(scale, ymin, ymax)[order(c(ymin, ymax))], ylim = range(scale, ymin, ymax)[order(c(ymin, ymax))], xlab = "DEFAULT SCALE", ylab = "NEW SCALE") ; par(yaxt = "s") ; axis(side = 2, at = scale)*

*# approximate number of main ticks, scale inversion*

*# ymin = 3.101 ; ymax = 2 ; n = 5 ; scale <- fun\_scale(n = n, lim = c(ymin, ymax), kind = "approx") ; scale ; par(yaxt = "n", yaxs = "i", las = 1) ; plot(ymin:ymax, ymin:ymax, xlim = range(scale, ymin, ymax)[order(c(ymin, ymax))], ylim = range(scale, ymin, ymax)[order(c(ymin, ymax))], xlab = "DEFAULT SCALE", ylab = "NEW SCALE") ; par(yaxt = "s") ; axis(side = 2, at = scale)*

*# DEBUGGING*

*# n = 9 ; lim = c(2, 3.101) ; kind = "approx" ; lib.path = NULL # for function debugging*

*# n = 10 ; lim = c(1e-4, 1e6) ; kind = "approx" ; lib.path = NULL # for function debugging*

*# function name*

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

*# end function name*

*# end initial argument checking*

*# required function checking*

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

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

**stop(paste0("\n\n================\n\n", tempo.cat, "\n\n================\n\n"), call. = FALSE)** *# == in stop() to be able to add several messages between ==*

**}**

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

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

**stop(paste0("\n\n================\n\n", tempo.cat, "\n\n================\n\n"), call. = FALSE)** *# == in stop() to be able to add several messages between ==*

**}**

*# end required function checking*

*# argument checking*

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

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

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

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

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

**if(tempo$problem == FALSE & isTRUE(all.equal(n, 0))){** *# isTRUE(all.equal(n, 0)) equivalent to n == 0 but deals with floats (approx ok)*

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

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

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

**}**

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

**if(tempo$problem == FALSE & all(diff(lim) == 0)){** *# isTRUE(all.equal(diff(lim), rep(0, length(diff(lim))))) not used because we strictly need zero as a result*

**tempo.cat <- paste0("ERROR IN ", function.name, ": lim ARGUMENT HAS A NULL RANGE (2 IDENTICAL VALUES)")**

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

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

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

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

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

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

**}**

**tempo <- fun\_check(data = kind, options = c("approx", "strict", "strict.cl"), length = 1, fun.name = function.name) ; eval(ee)**

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

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

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

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

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

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

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

**}**

**}**

**}**

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

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

**}**

*# end argument checking with fun\_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\_check\_dev)) # activate this line and use the function (with no arguments left as NULL) to check arguments status and if they have been checked using fun\_check()*

*# end argument checking*

*# main code*

**lim.rank <- rank(lim)** *# to deal with inverted axis*

**lim <- sort(lim)**

**if(kind == "approx"){**

*# package checking*

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

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

*# end package checking*

**output <- ggplot2::ggplot\_build(ggplot2::ggplot() + ggplot2::scale\_y\_continuous(**

**breaks = scales::trans\_breaks(**

**trans = "identity",**

**inv = "identity",**

**n = n**

**),**

**limits = lim**

**))$layout$panel\_params[[1]]$y$breaks** *# pretty() alone is not appropriate: tempo.pret <- pretty(seq(lim[1] ,lim[2], length.out = n)) ; tempo.pret[tempo.pret > = lim[1] & tempo.pret < = lim[2]]. # in ggplot 3.3.0, tempo.coord$y.major\_source replaced by tempo.coord$y$breaks*

**if( ! is.null(attributes(output))){** *# layout$panel\_params[[1]]$y$breaks can be characters (labels of the axis). In that case, it has attributes that corresponds to positions*

**output <- unlist(attributes(output))**

**}**

**output <- output[ ! is.na(output)]**

**}else if(kind == "strict"){**

**output <- fun\_round(seq(lim[1] ,lim[2], length.out = n), 2)**

**}else if(kind == "strict.cl"){**

**tempo.range <- diff(sort(lim))**

**tempo.max <- max(lim)**

**tempo.min <- min(lim)**

**mid <- tempo.min + (tempo.range/2)** *# middle of axis*

**tempo.inter <- tempo.range / (n + 1)** *# current interval between two ticks, between 0 and Inf*

**if(tempo.inter == 0){** *# isTRUE(all.equal(tempo.inter, rep(0, length(tempo.inter)))) not used because we strictly need zero as a result*

**tempo.cat <- paste0("ERROR IN ", function.name, ": THE INTERVAL BETWEEN TWO TICKS OF THE SCALE IS NULL. MODIFY THE lim OR n ARGUMENT")**

**stop(paste0("\n\n================\n\n", tempo.cat, "\n\n================\n\n"), call. = FALSE)** *# == in stop() to be able to add several messages between ==*

**}**

**log10.abs.lim <- 200**

**log10.range <- (-log10.abs.lim):log10.abs.lim**

**log10.vec <- 10^log10.range**

**round.vec <- c(5, 4, 3, 2.5, 2, 1.25, 1)**

**dec.table <- outer(log10.vec, round.vec)** *# table containing the scale units (row: power of ten from -201 to +199, column: the 5, 2.5, 2, 1.25, 1 notches*



*# recover the number of leading zeros in tempo.inter*

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

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

**if(any(grepl(pattern = "\\+", x = tempo.inter))){ # tempo.inter > 1**

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

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

**}else if(any(grepl(pattern = "\\-", x = tempo.inter))){ # tempo.inter < 1**

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

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

**}else{**

**tempo.cat <- paste0("ERROR IN ", function.name, ": CODE INCONSISTENCY 1")**

**stop(paste0("\n\n================\n\n", tempo.cat, "\n\n================\n\n"), call. = FALSE)** *# == in stop() to be able to add several messages between ==*

**}**

**tempo.scale <- dec.table[log10.range == power10.exp, ]**

*# new interval*

**inter.select <- NULL**

**for(i1 in 1:length(tempo.scale)){**

**tempo.first.tick <- trunc((tempo.min + tempo.scale[i1]) / tempo.scale[i1]) \* (tempo.scale[i1])** *# this would be use to have a number not multiple of tempo.scale[i1]: ceiling(tempo.min) + tempo.scale[i1] \* 10^power10.exp*

**tempo.last.tick <- tempo.first.tick + tempo.scale[i1] \* (n - 1)**

**if((tempo.first.tick >= tempo.min) & (tempo.last.tick <= tempo.max)){**

**inter.select <- tempo.scale[i1]**

**break()**

**}**

**}**

**if(is.null(inter.select)){**

**tempo.cat <- paste0("ERROR IN ", function.name, ": CODE INCONSISTENCY 2")**

**stop(paste0("\n\n================\n\n", tempo.cat, "\n\n================\n\n"), call. = FALSE)** *# == in stop() to be able to add several messages between ==*

**}**

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

*# end new interval*

*# centering the new scale*

**tempo.mid <- trunc((mid + (-1:1) \* inter.select) / inter.select) \* inter.select** *# tempo middle tick closest to the middle axis*

**mid.tick <- tempo.mid[which.min(abs(tempo.mid - mid))]**

**if(isTRUE(all.equal(n, rep(1, length(n))))){** *# isTRUE(all.equal(n, rep(1, length(n)))) is similar to n == 1 but deals with float*

**output <- mid.tick**

**}else if(isTRUE(all.equal(n, rep(2, length(n))))){** *# isTRUE(all.equal(n, rep(0, length(n)))) is similar to n == 2 but deals with float*

**output <- mid.tick**

**tempo.min.dist <- mid.tick - inter.select - tempo.min**

**tempo.max.dist <- tempo.max - mid.tick + inter.select**

**if(tempo.min.dist <= tempo.max.dist){** *# distance between lowest tick and bottom axis <= distance between highest tick and top axis. If yes, extra tick but at the top, otherwise at the bottom*

**output <- c(mid.tick, mid.tick + inter.select)**

**}else{**

**output <- c(mid.tick - inter.select, mid.tick)**

**}**

**}else if((n / 2 - trunc(n / 2)) > 0.1){** *# > 0.1 to avoid floating point. Because result can only be 0 or 0.5. Thus, > 0.1 means odd number*

**output <- c(mid.tick - (trunc(n / 2):1) \* inter.select, mid.tick, mid.tick + (1:trunc(n / 2)) \* inter.select)**

**}else if((n / 2 - trunc(n / 2)) < 0.1){** *# < 0.1 to avoid floating point. Because result can only be 0 or 0.5. Thus, < 0.1 means even number*

**tempo.min.dist <- mid.tick - trunc(n / 2) \* inter.select - tempo.min**

**tempo.max.dist <- tempo.max - mid.tick + trunc(n / 2) \* inter.select**

**if(tempo.min.dist <= tempo.max.dist){** *# distance between lowest tick and bottom axis <= distance between highest tick and top axis. If yes, extra tick but at the bottom, otherwise at the top*

**output <- c(mid.tick - ((trunc(n / 2) - 1):1) \* inter.select, mid.tick, mid.tick + (1:trunc(n / 2)) \* inter.select)**

**}else{**

**output <- c(mid.tick - (trunc(n / 2):1) \* inter.select, mid.tick, mid.tick + (1:(trunc(n / 2) - 1)) \* inter.select)**

**}**

**}else{**

**tempo.cat <- paste0("ERROR IN ", function.name, ": CODE INCONSISTENCY 3")**

**stop(paste0("\n\n================\n\n", tempo.cat, "\n\n================\n\n"), call. = FALSE)** *# == in stop() to be able to add several messages between ==*

**}**

*# end centering the new scale*

*# last check*

**if(min(output) < tempo.min){**

**output <- c(output[-1], max(output) + inter.select)** *# remove the lowest tick and add a tick at the top*

**}else if( max(output) > tempo.max){**

**output <- c(min(output) - inter.select, output[-length(output)])**

**}**

**if(min(output) < tempo.min | max(output) > tempo.max){**

**tempo.cat <- paste0("ERROR IN ", function.name, ": CODE INCONSISTENCY 4")**

**stop(paste0("\n\n================\n\n", tempo.cat, "\n\n================\n\n"), call. = FALSE)** *# == in stop() to be able to add several messages between ==*

**}**

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

**tempo.cat <- paste0("ERROR IN ", function.name, ": CODE INCONSISTENCY 5 (NA GENERATION)")**

**stop(paste0("\n\n================\n\n", tempo.cat, "\n\n================\n\n"), call. = FALSE)** *# == in stop() to be able to add several messages between ==*

**}**

*# end last check*

**}else{**

**tempo.cat <- paste0("ERROR IN ", function.name, ": CODE INCONSISTENCY 6")**

**stop(paste0("\n\n================\n\n", tempo.cat, "\n\n================\n\n"), call. = FALSE)** *# == in stop() to be able to add several messages between ==*

**}**

**if(diff(lim.rank) < 0){**

**output <- rev(output)**

**}**

**return(output)**

**}**

######## fun\_inter\_ticks() #### define coordinates of secondary ticks

**fun\_inter\_ticks <- function(lim, log = "log10", breaks = NULL, n = NULL, warn.print = TRUE){**

*# AIM*

*# define coordinates and values of secondary ticks*

*# ARGUMENTS*

*# lim: vector of 2 numbers indicating the limit range of the axis. Order of the 2 values matters (for inverted axis). If log argument is "log2" or "log10", values in lim must be already log transformed. Thus, negative or zero values are allowed*

*# log: either "log2" (values in the lim argument are log2 transformed) or "log10" (values in the lim argument are log10 transformed), or "no"*

*# breaks: mandatory vector of numbers indicating the main ticks values/positions when log argument is "no". Ignored when log argument is "log2" or "log10"*

*# n: number of secondary ticks between each main tick when log argument is "no". Ignored when log argument is "log2" or "log10"*

*# warn.print: logical. Print potential warning messages at the end of the execution? If FALSE, warning messages are never printed, but can still be recovered in the returned list*

*# REQUIRED PACKAGES*

*# none*

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

*# fun\_check()*

*# RETURN*

*# a list containing*

*# $log: value of the log argument used*

*# $coordinates: the coordinates of the secondary ticks on the axis, between the lim values*

*# $values: the corresponding values associated to each coordinate (with log scale, 2^$values or 10^$values is equivalent to the labels of the axis)*

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

*# EXAMPLES*

*# no log scale*

*# fun\_inter\_ticks(lim = c(-4,4), log = "no", breaks = c(-2, 0, 2), n = 3)*

*# fun\_inter\_ticks(lim = c(10, 0), log = "no", breaks = c(10, 8, 6, 4, 2, 0), n = 4)*

*# log2*

*# fun\_inter\_ticks(lim = c(-4,4), log = "log2")*

*# log10*

*# fun\_inter\_ticks(lim = c(-2,3), log = "log10")*

*# DEBUGGING*

*# lim = c(2, 3.101) ; log = "no" ; breaks = NULL ; n = NULL ; warn.print = TRUE # for function debugging*

*# lim = c(0, 26.5) ; log = "no" ; breaks = c(0, 10, 20) ; n = 3 # for function debugging*

*# lim = c(10, 0); log = "no"; breaks = c(10, 8, 6, 4, 2, 0); n = 4 # for function debugging*

*# lim = c(-10, -20); log = "no"; breaks = c(-20, -15, -10); n = 4 # for function debugging*

*# function name*

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

*# end function name*

*# required function checking*

**req.function <- c(**

**"fun\_check"**

**)**

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

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

**tempo.cat <- paste0("ERROR IN ", function.name, "\nREQUIRED ", i1, "() FUNCTION IS MISSING IN THE R ENVIRONMENT")**

**stop(paste0("\n\n================\n\n", tempo.cat, "\n\n================\n\n"), call. = FALSE)** *# == in stop() to be able to add several messages between ==*

**}**

**}**

*# end required function checking*

*# argument primary checking*

*# arg with no default values*

**if(any(missing(lim))){**

**tempo.cat <- paste0("ERROR IN ", function.name, "\nARGUMENT lim HAS NO DEFAULT VALUE AND REQUIRES ONE")**

**stop(paste0("\n\n================\n\n", tempo.cat, "\n\n================\n\n"), call. = FALSE)** *# == in stop() to be able to add several messages between ==*

**}**

*# end arg with no default values*

*# using fun\_check()*

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

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

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

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

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

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

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

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

**}**

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

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

**}**

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

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

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

**}**

*# end using fun\_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\_check\_dev)) # activate this line and use the function (with no arguments left as NULL) to check arguments status and if they have been checked using fun\_check()*

*# end argument primary checking*

*# second round of checking and data preparation*

*# dealing with NA*

**if(any(is.na(lim)) | any(is.na(log)) | any(is.na(breaks)) | any(is.na(n)) | any(is.na(warn.print))){**

**tempo.cat <- paste0("ERROR IN ", function.name, "\nNO ARGUMENT CAN HAVE NA VALUES")**

**stop(paste0("\n\n================\n\n", tempo.cat, "\n\n================\n\n"), call. = FALSE)** *# == in stop() to be able to add several messages between ==*

**}**

*# end dealing with NA*

*# dealing with NULL*

**if(is.null(lim) | is.null(log)){**

**tempo.cat <- paste0("ERROR IN ", function.name, "\nTHESE ARGUMENTS\nlim\nlog\nCANNOT BE NULL")**

**stop(paste0("\n\n================\n\n", tempo.cat, "\n\n================\n\n"), call. = FALSE)** *# == in stop() to be able to add several messages between ==*

**}**

*# end dealing with NULL*

**if(all(diff(lim) == 0)){** *# isTRUE(all.equal(diff(lim), rep(0, length(diff(lim))))) not used because we strictly need zero as a result*

**tempo.cat <- paste0("ERROR IN ", function.name, "\nlim ARGUMENT HAS A NULL RANGE (2 IDENTICAL VALUES): ", paste(lim, collapse = " "))**

**stop(paste0("\n\n================\n\n", tempo.cat, "\n\n================\n\n"), call. = FALSE)** *# == in stop() to be able to add several messages between ==*

**}else if(any(lim %in% c(Inf, -Inf))){**

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

**stop(paste0("\n\n================\n\n", tempo.cat, "\n\n================\n\n"), call. = FALSE)** *# == in stop() to be able to add several messages between ==*

**}**

**if(log == "no" & is.null(breaks)){**

**tempo.cat <- paste0("ERROR IN ", function.name, "\nbreaks ARGUMENT CANNOT BE NULL IF log ARGUMENT IS \"no\"")**

**stop(paste0("\n\n================\n\n", tempo.cat, "\n\n================\n\n"), call. = FALSE)** *# == in stop() to be able to add several messages between ==*

**}**

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

**if(length(breaks) < 2){**

**tempo.cat <- paste0("ERROR IN ", function.name, "\nbreaks ARGUMENT MUST HAVE 2 VALUES AT LEAST (OTHERWISE, INTER TICK POSITIONS CANNOT BE COMPUTED): ", paste(breaks, collapse = " "))**

**stop(paste0("\n\n================\n\n", tempo.cat, "\n\n================\n\n"), call. = FALSE)** *# == in stop() to be able to add several messages between ==*

**}**

**if( ! isTRUE(all.equal(diff(sort(breaks)), rep(diff(sort(breaks))[1], length(diff(sort(breaks))))))){ #** *isTRUE(all.equal(n, 0)) equivalent to n == 0 but deals with floats (approx ok)*

**tempo.cat <- paste0("ERROR IN ", function.name, "\nbreaks ARGUMENT MUST HAVE EQUIDISTANT VALUES (OTHERWISE, EQUAL NUMBER OF INTER TICK BETWEEN MAIN TICKS CANNOT BE COMPUTED): ", paste(breaks, collapse = " "))**

**stop(paste0("\n\n================\n\n", tempo.cat, "\n\n================\n\n"), call. = FALSE)** *# == in stop() to be able to add several messages between ==*

**}**

**}**

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

**if(n <= 0){**

**tempo.cat <- paste0("ERROR IN ", function.name, "\nn ARGUMENT MUST BE A POSITIVE AND NON NULL INTEGER: ", paste(n, collapse = " "))**

**stop(paste0("\n\n================\n\n", tempo.cat, "\n\n================\n\n"), call. = FALSE)** *# == in stop() to be able to add several messages between ==*

**}**

**}**

*# end second round of checking and data preparation*

*# main code*

**ini.warning.length <- options()$warning.length**

**options(warning.length = 8170)**

**on.exit(exp = options(warning.length = ini.warning.length))**

**warn <- NULL**

**warn.count <- 0**

**lim.rank <- rank(lim)** *# to deal with inverse axis*

**if(log != "no"){**

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

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

**power10.exp <- as.integer(substring(text = 10^lim, first = (regexpr(pattern = "\\+|\\-", text = 10^lim))))** *# recover the power of 10, i.e., integer part of lim. Example recover 08 from 1e+08. Works for log2*

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

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

**tick.pos <- unique(as.vector(outer(2:10, ifelse(log == "log2", 2, 10)^((power10.exp[1] - ifelse(diff(lim.rank) > 0, 1, -1)):(power10.exp[2] + ifelse(diff(lim.rank) > 0, 1, -1))))))** *# use log10(2:10) even if log2: it is to get log values between 0 and 1*

**tick.pos <- sort(tick.pos, decreasing = ifelse(diff(lim.rank) > 0, FALSE, TRUE))**

**if(log == "log2"){**

**tick.values <- tick.pos[tick.pos >= min(2^lim) & tick.pos <= max(2^lim)]**

**tick.pos <- log2(tick.values)**

**}else if(log == "log10"){**

**tick.values <- tick.pos[tick.pos >= min(10^lim) & tick.pos <= max(10^lim)]**

**tick.pos <- log10(tick.values)**

**}**

**}else{**

*# if(length(breaks) > 1){ # not required because already checked above*

**breaks.rank <- rank(c(breaks[1], breaks[length(breaks)]))**

**if(diff(breaks.rank) != diff(lim.rank)){**

**breaks <- sort(breaks, decreasing = ifelse(diff(lim.rank) < 0, TRUE, FALSE))**

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

**tempo.warn <- paste0("(", warn.count,") VALUES IN breaks ARGUMENT NOT IN THE SAME ORDER AS IN lim ARGUMENT -> VALUES REORDERED AS IN lim: ", paste(breaks, collapse = " "))**

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

**breaks.rank <- rank(c(breaks[1], breaks[length(breaks)]))**

**}**

*# }*

**main.tick.dist <- mean(diff(breaks), na.rm = TRUE)**

**tick.dist <- main.tick.dist / (n + 1)**

**tempo.extra.margin <- max(abs(diff(breaks)), na.rm = TRUE)**

**tick.pos <- seq(**

**if(diff(breaks.rank) > 0){breaks[1] - tempo.extra.margin}else{breaks[1] + tempo.extra.margin},**

**if(diff(breaks.rank) > 0){breaks[length(breaks)] + tempo.extra.margin}else{breaks[length(breaks)] - tempo.extra.margin},**

**by = tick.dist**

**)**

**tick.pos <- tick.pos[tick.pos >= min(lim) & tick.pos <= max(lim)]**

**tick.values <- tick.pos**

**}**

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

**tempo.cat <- paste0("INTERNAL CODE ERROR IN ", function.name, ": NA or Inf GENERATED FOR THE INTER TICK POSITIONS: ", paste(tick.pos, collapse = " "))**

**stop(paste0("\n\n================\n\n", tempo.cat, "\n\n================\n\n", ifelse(is.null(warn), "", paste0("IN ADDITION\nWARNING", ifelse(warn.count > 1, "S", ""), ":\n\n", warn))), call. = FALSE)** *# == in stop() to be able to add several messages between ==*

**}**

**if(length(tick.pos) == 0){**

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

**tempo.warn <- paste0("(", warn.count,") NO INTER TICKS COMPUTED BETWEEN THEN LIMITS INDICATED: ", paste(lim, collapse = " "))**

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

**}**

**output <- list(log = log, coordinates = tick.pos, values = tick.values, warn = warn)**

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

**warning(paste0("FROM ", function.name, ":\n\n", warn), call. = FALSE)** *# to recover the warning messages, see $warn*

**}**

**return(output)**

**}**

######## fun\_post\_plot() #### set graph param after plotting (axes redesign for instance)

****

**# 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.size = 1.5, x.label.size = 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.size = 1.5, y.label.size = 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 = "", corner.text.size = 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\_check()*

*# fun\_open() 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.size: positive numeric. Increase or decrease the size of the x axis numbers. Value 1 does not change it, 0.5 decreases by half, 2 increases by 2. Also control the size of displayed categories*

*# x.label.size: positive numeric. Increase or decrease the size of the x axis legend text. Value 1 does not change it, 0.5 decreases by half, 2 increases by 2*

*# 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.size: positive numeric. Increase or decrease the size of the y axis numbers. Value 1 does not change it, 0.5 decreases by half, 2 increases by 2. Also control the size of displayed categories*

*# y.label.size: positive numeric. Increase or decrease the size of the y axis legend text. Value 1 does not change it, 0.5 decreases by half, 2 increases by 2*

*# 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*

*# corner.text.size: positive numeric. Increase or decrease the size of the text. Value 1 does not change it, 0.5 decreases by half, 2 increases by 2*

*# 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$ylog, x.lab = "Values", y.lab = "TEST", y.axis.size = 1.25, y.label.size = 1.5, y.dist.legend = 0.7, just.label.add = ! prior.par$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.size = 1, y.label.size = 2, y.dist.legend = 0.6)*

*# Example of title easily added to a plot:*

*# plot(1:100) ; para <- fun\_post\_plot(corner.text = "TITLE ADDED") # try also: par(xpd = TRUE) ; text(x = para$x.mid.left.fig.region, y = para$y.mid.top.fig.region, labels = "TITLE ADDED", cex = 0.5)*

*# example with margins in the device region:*

*# windows(5,5) ; fun\_prior\_plot(box.type = "o") ; 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.size = 1.5 ; x.label.size = 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.size = 1.5 ; y.label.size = 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 = "" ; corner.text.size = 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(utils::find("fun\_check", mode = "function")) == 0){**

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

**stop(paste0("\n\n================\n\n", tempo.cat, "\n\n================\n\n"), call. = FALSE)** *# == in stop() to be able to add several messages between ==*

**}**

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

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

**stop(paste0("\n\n================\n\n", tempo.cat, "\n\n================\n\n"), call. = FALSE)** *# == in stop() to be able to add several messages between ==*

**}**

*# end required function checking*

*# argument checking*

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

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

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

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

**tempo <- fun\_check(data = x.side, options = c(0, 1, 3), length = 1, fun.name = function.name) ; eval(ee)**

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

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

**tempo <- fun\_check(data = x.categ, class = "character", na.contain = TRUE, fun.name = function.name) ; eval(ee)**

**}**

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

**tempo <- fun\_check(data = x.categ.pos, class = "vector", mode = "numeric", fun.name = function.name) ; eval(ee)**

**}**

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

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

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

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

**tempo <- fun\_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\_check(data = y.side, options = c(0, 2, 4), length = 1, fun.name = function.name) ; eval(ee)**

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

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

**tempo <- fun\_check(data = y.categ, class = "character", na.contain = TRUE, fun.name = function.name) ; eval(ee)**

**}**

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

**tempo <- fun\_check(data = y.categ.pos, class = "vector", mode = "numeric", fun.name = function.name) ; eval(ee)**

**}**

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

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

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

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

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

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

**tempo <- fun\_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\_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("ERROR IN ", function.name, ": bg.color ARGUMENT MUST BE A HEXADECIMAL COLOR VECTOR STARTING BY # OR A COLOR NAME GIVEN BY colors()")**

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

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

**}**

**}**

**if( ! is.null(grid.lwd)){**

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

**}**

**if( ! is.null(grid.col)){**

**tempo <- fun\_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("ERROR IN ", function.name, ": grid.col ARGUMENT MUST BE A HEXADECIMAL COLOR VECTOR STARTING BY # OR A COLOR NAME GIVEN BY colors()")**

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

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

**}**

**}**

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

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

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

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

**if( ! is.null(custom.par)){**

**tempo <- fun\_check(data = custom.par, typeof = "list", length = 1, fun.name = function.name) ; eval(ee)**

**}**

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

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

**}**

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

*# end argument checking*

*# 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(any(sapply(FUN = all.equal, c(1, 3), x.side) == TRUE)){**

**par(xpd=FALSE, xaxt="s")**

**if(is.null(x.categ) & x.log.scale == TRUE){**

**if(any(par()$xaxp[1:2] == 0)){** *# any(sapply(FUN = all.equal, par()$xaxp[1:2], 0) == TRUE) not used because we strictly need zero as a result. Beware: write "== TRUE", because the result is otherwise character and a warning message appears using any()*

**if(par()$xaxp[1] == 0){** *# isTRUE(all.equal(par()$xaxp[1], 0)) not used because we strictly need zero as a result*

**par(xaxp = c(10^-30, par()$xaxp[2:3]))** *# because log10(par()$xaxp[1] == 0) == -Inf*

**}**

**if(par()$xaxp[2] == 0){** *# isTRUE(all.equal(par()$xaxp[1], 0)) not used because we strictly need zero as a result*

**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.size)**

**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.size)**

**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()$xaxp[3]+1), 2), cex.axis = x.axis.size)** *# axis(side=x.side, at=round(seq(par()$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.size) # to get the 1000 comma separator*

**mtext(side = x.side, text = x.lab, line = x.dist.legend / 0.2, las = 0, cex = x.label.size)**

**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)){**

**tempo.cat <- paste0("ERROR IN ", function.name, ": x.categ.pos MUST BE THE SAME LENGTH AS x.categ")**

**stop(paste0("\n\n================\n\n", tempo.cat, "\n\n================\n\n"), call. = FALSE)** *# == in stop() to be able to add several messages between ==*

**}**

**par(xpd = TRUE)**

**if(isTRUE(all.equal(x.side, 1))){** *#isTRUE(all.equal(x.side, 1)) is similar to x.side == 1 but deals with float*

**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.size)**

**}else if(isTRUE(all.equal(x.side, 3))){** *#isTRUE(all.equal(x.side, 1)) is similar to x.side == 3 but deals with float*

**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.size)**

**}else{**

**tempo.cat <- paste0("ERROR IN ", function.name, ": ARGUMENT x.side CAN ONLY BE 1 OR 3")**

**stop(paste0("\n\n================\n\n", tempo.cat, "\n\n================\n\n"), call. = FALSE)** *# == in stop() to be able to add several messages between ==*

**}**

**par(xpd = FALSE)**

**x.text <- par("usr")[2]**

**}else{**

**tempo.cat <- paste0("ERROR IN ", function.name, ": PROBLEM WITH THE x.side (", x.side ,") OR x.log.scale (", x.log.scale,") ARGUMENTS")**

**stop(paste0("\n\n================\n\n", tempo.cat, "\n\n================\n\n"), call. = FALSE)** *# == in stop() to be able to add several messages between ==*

**}**

**}else{**

**x.text <- par("usr")[2]**

**}**

**if(any(sapply(FUN = all.equal, c(2, 4), y.side) == TRUE)){**

**par(xpd=FALSE, yaxt="s")**

**if(is.null(y.categ) & y.log.scale == TRUE){**

**if(any(par()$yaxp[1:2] == 0)){** *# any(sapply(FUN = all.equal, par()$yaxp[1:2], 0) == TRUE) not used because we strictly need zero as a result. Beware: write "== TRUE", because the result is otherwise character and a warning message appears using any()*

**if(par()$yaxp[1] == 0){** *# strict zero needed*

**par(yaxp = c(10^-30, par()$yaxp[2:3]))** *# because log10(par()$yaxp[1] == 0) == -Inf*

**}**

**if(par()$yaxp[2] == 0){** *# strict zero needed*

**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.size)**

**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.size)**

**}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.size)**

**mtext(side = y.side, text = y.lab, line = y.dist.legend / 0.2, las = 0, cex = y.label.size)**

**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)){**

**tempo.cat <- paste0("ERROR IN ", function.name, ": y.categ.pos MUST BE THE SAME LENGTH AS y.categ")**

**stop(paste0("\n\n================\n\n", tempo.cat, "\n\n================\n\n"), call. = FALSE)** *# == in stop() to be able to add several messages between ==*

**}**

**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(isTRUE(all.equal(y.side, 2))){** *#isTRUE(all.equal(y.side, 2)) is similar to y.side == 2 but deals with float*

**text(x = x.mid.left.fig.region, y = y.categ.pos, labels = y.categ, srt = text.angle, cex = y.axis.size)**

**}else if(isTRUE(all.equal(y.side, 4))){ # idem**

**text(x = x.mid.right.fig.region, y = y.categ.pos, labels = y.categ, srt = text.angle, cex = y.axis.size)**

**}else{**

**tempo.cat <- paste0("ERROR IN ", function.name, ": ARGUMENT y.side CAN ONLY BE 2 OR 4")**

**stop(paste0("\n\n================\n\n", tempo.cat, "\n\n================\n\n"), call. = FALSE)** *# == in stop() to be able to add several messages between ==*

**}**

**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{**

**tempo.cat <- paste0("ERROR IN ", function.name, ": PROBLEM WITH THE y.side (", y.side ,") OR y.log.scale (", y.log.scale,") ARGUMENTS")**

**stop(paste0("\n\n================\n\n", tempo.cat, "\n\n================\n\n"), call. = FALSE)** *# == in stop() to be able to add several messages between ==*

**}**

**}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 = corner.text.size)** *# 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 & isTRUE(all.equal(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.size)** *# x label*

**}**

**if(just.label.add == TRUE & isTRUE(all.equal(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.size)** *# x label*

**}**

**par(xpd=FALSE)**

**if(par.reset == TRUE){**

**tempo.par <- fun\_open(pdf = 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("ERROR IN ", function.name, ": custom.par ARGUMENT SHOULD HAVE THE NAMES OF THE COMPARTMENT LIST COMING FROM THE par() LIST")**

**stop(paste0("\n\n================\n\n", tempo.cat, "\n\n================\n\n"), call. = FALSE)** *# == in stop() to be able to add several messages between ==*

**}**

**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() #### close specific graphic windows

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

**fun\_close <- function(kind = "pdf", return.text = FALSE){**

*# AIM*

*# close only specific graphic windows (devices)*

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

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

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

**stop(paste0("\n\n================\n\n", tempo.cat, "\n\n================\n\n"), call. = FALSE)** *# == in stop() to be able to add several messages between ==*

**}**

*# end required function checking*

*# argument checking*

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

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

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

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

**tempo <- fun\_check(data = kind, options = c("windows", "quartz", "x11", "X11", "pdf", "bmp", "png", "tiff"), fun.name = function.name) ; eval(ee)**

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

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

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

**}**

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

*# end argument checking*

*# 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 = NULL, text.size = 1, title = NULL, title.size = 1.5){**

*# 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*

*# title: character string of the graph title*

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

*# REQUIRED PACKAGES*

*# none*

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

*# fun\_check()*

*# RETURN*

*# an empty plot*

*# EXAMPLES*

*# simple example*

*# fun\_empty\_graph(text = "NO GRAPH")*

*# white page*

*# fun\_empty\_graph() # white page*

*# all the arguments*

*# fun\_empty\_graph(text = "NO GRAPH", text.size = 2, title = "GRAPH1", title.size = 1)*

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

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

**stop(paste0("\n\n================\n\n", tempo.cat, "\n\n================\n\n"), call. = FALSE)** *# == in stop() to be able to add several messages between ==*

**}**

*# end required function checking*

*# argument checking*

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

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

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

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

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

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

**}**

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

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

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

**}**

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

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

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

**}**

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

*# end argument checking*

*# 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]))**

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

**text(x = 1, y = 1, labels = text, cex = text.size)**

**}**

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

**text(x = x.left.dev.region, y = y.top.dev.region, labels = title, adj=c(0, 1), cex = title.size)**

**}**

**par(ini.par)**

**}**

################ gg graphics

######## fun\_gg\_palette() #### ggplot2 default color palette



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

**fun\_gg\_palette <- function(n, kind = "std"){**

*# 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*

*# kind: either "std" for standard gg colors, "dark" for darkened gg colors, or "light" for pastel gg colors*

*# REQUIRED PACKAGES*

*# none*

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

*# fun\_check()*

*# RETURN*

*# the vector of hexadecimal colors*

*# EXAMPLES*

*# output of the function*

*# fun\_gg\_palette(n = 2)*

*# the ggplot2 palette when asking for 7 different colors*

*# plot(1:7, pch = 16, cex = 5, col = fun\_gg\_palette(n = 7))*

*# selection of the 5th color of the ggplot2 palette made of 7 different colors*

*# plot(1:7, pch = 16, cex = 5, col = fun\_gg\_palette(n = 7)[5])*

*# the ggplot2 palette made of 7 darkened colors*

*# plot(1:7, pch = 16, cex = 5, col = fun\_gg\_palette(n = 7, kind = "dark"))*

*# the ggplot2 palette made of 7 lighten colors*

*# plot(1:7, pch = 16, cex = 5, col = fun\_gg\_palette(n = 7, kind = "light"))*

*# DEBUGGING*

*# n = 0*

*# kind = "std"*

*# function name*

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

*# end function name*

*# required function checking*

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

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

**stop(paste0("\n\n================\n\n", tempo.cat, "\n\n================\n\n"), call. = FALSE)** *# == in stop() to be able to add several messages between ==*

**}**

*# end required function checking*

*# argument checking*

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

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

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

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

**tempo <- fun\_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 & isTRUE(all.equal(n, 0))){** *# isTRUE(all.equal(n, 0))) is similar to n == 0 but deals with float*

**tempo.cat <- paste0("ERROR IN ", function.name, ": n ARGUMENT MUST BE A NON ZERO INTEGER. HERE IT IS: ", paste(n, collapse = " "))**

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

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

**tempo <- fun\_check(data = kind, options = c("std", "dark", "light"), length = 1, fun.name = function.name) ; eval(ee)**

**}**

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

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

**}**

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

*# end argument checking*

*# main code*

**hues = seq(15, 375, length = n + 1)**

**hcl(h = hues, l = if(kind == "std"){65}else if(kind == "dark"){35}else if(kind == "light"){85}, c = 100)[1:n]**

**}**

######## fun\_gg\_just() #### ggplot2 justification of the axis labeling, depending on angle



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

**fun\_gg\_just <- function(angle, axis){**

*# AIM*

*# provide correct justification for axis labeling, depending on the chosen angle*

*# ARGUMENTS*

*# angle: integer value of the text angle for the axis labels, using the same rules as in ggplot2. Positive values for counterclockwise rotation: 0 for horizontal, 90 for vertical, 180 for upside down etc. Negative values for clockwise rotation: 0 for horizontal, -90 for vertical, -180 for upside down etc.*

*# axis: which axis for? Either "x" or "y"*

*# REQUIRED PACKAGES*

*# none*

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

*# fun\_check()*

*# RETURN*

*# a list containing:*

*# $angle: the submitted angle (value potentially reduced to fit the [-360 ; 360] interval, e.g., 460 -> 100, without impact on the final angle displayed)*

*# $hjust: the horizontal justification*

*# $vjust: the vertical justification*

*# EXAMPLES*

*# fun\_gg\_just(angle = 45, axis = "x")*

*# fun\_gg\_just(angle = (360\*2 + 45), axis = "y")*

*# output <- fun\_gg\_just(angle = 45, axis = "x") ; obs1 <- data.frame(time = 1:20, group = rep(c("CLASS\_1", "CLASS\_2"), times = 10)) ; ggplot2::ggplot() + ggplot2::geom\_bar(data = obs1, mapping = ggplot2::aes(x = group, y = time), stat = "identity") + ggplot2::theme(axis.text.x = ggplot2::element\_text(angle = output$angle, hjust = output$hjust, vjust = output$vjust))*

*# output <- fun\_gg\_just(angle = -45, axis = "y") ; obs1 <- data.frame(time = 1:20, group = rep(c("CLASS\_1", "CLASS\_2"), times = 10)) ; ggplot2::ggplot() + ggplot2::geom\_bar(data = obs1, mapping = ggplot2::aes(x = group, y = time), stat = "identity") + ggplot2::theme(axis.text.y = ggplot2::element\_text(angle = output$angle, hjust = output$hjust, vjust = output$vjust)) + ggplot2::coord\_flip()*

*# output1 <- fun\_gg\_just(angle = 90, axis = "x") ; output2 <- fun\_gg\_just(angle = -45, axis = "y") ; obs1 <- data.frame(time = 1:20, group = rep(c("CLASS\_1", "CLASS\_2"), times = 10)) ; ggplot2::ggplot() + ggplot2::geom\_bar(data = obs1, mapping = ggplot2::aes(x = group, y = time), stat = "identity") + ggplot2::theme(axis.text.x = ggplot2::element\_text(angle = output1$angle, hjust = output1$hjust, vjust = output1$vjust), axis.text.y = ggplot2::element\_text(angle = output2$angle, hjust = output2$hjust, vjust = output2$vjust))*

*# DEBUGGING*

*# angle = 45 ; axis = "y"*

*# function name*

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

*# end function name*

*# required function checking*

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

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

**stop(paste0("\n\n================\n\n", tempo.cat, "\n\n================\n\n"), call. = FALSE)** *# == in stop() to be able to add several messages between ==*

**}**

*# end required function checking*

*# argument checking*

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

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

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

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

**tempo <- fun\_check(data = angle, class = "integer", length = 1, double.as.integer.allowed = TRUE, neg.values = TRUE, fun.name = function.name) ; eval(ee)**

**tempo <- fun\_check(data = axis, options = c("x", "y"), length = 1, fun.name = function.name) ; eval(ee)**

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

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

**}**

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

*# end argument checking*

*# main code*

*# to get angle between -360 and 360*

**while(angle > 360){**

**angle <- angle - 360**

**}**

**while(angle < -360){**

**angle <- angle + 360**

**}**

*# end to get angle between -360 and 360*

*# justifications*

**if(axis == "x"){**

**if(any(sapply(FUN = all.equal, c(-360, -180, 0, 180, 360), angle) == TRUE)){** *# equivalent of angle == -360 | angle == -180 | angle == 0 | angle == 180 | angle == 360 but deals with floats*

**hjust <- 0.5**

**vjust <- 0.5**

**}else if(any(sapply(FUN = all.equal, c(-270, 90), angle) == TRUE)){**

**hjust <- 1**

**vjust <- 0.5**

**}else if(any(sapply(FUN = all.equal, c(-90, 270), angle) == TRUE)){**

**hjust <- 0**

**vjust <- 0.5**

**}else if((angle > -360 & angle < -270) | (angle > 0 & angle < 90)){**

**hjust <- 1**

**vjust <- 1**

**}else if((angle > -270 & angle < -180) | (angle > 90 & angle < 180)){**

**hjust <- 1**

**vjust <- 0**

**}else if((angle > -180 & angle < -90) | (angle > 180 & angle < 270)){**

**hjust <- 0**

**vjust <- 0**

**}else if((angle > -90 & angle < 0) | (angle > 270 & angle < 360)){**

**hjust <- 0**

**vjust <- 1**

**}**

**}else if(axis == "y"){**

**if(any(sapply(FUN = all.equal, c(-270, -90, 90, 270), angle) == TRUE)){** *# equivalent of angle == -270 | angle == -90 | angle == 90 | angle == 270 but deals with floats*

**hjust <- 0.5**

**vjust <- 0.5**

**}else if(any(sapply(FUN = all.equal, c(-360, 0, 360), angle) == TRUE)){**

**hjust <- 1**

**vjust <- 0.5**

**}else if(any(sapply(FUN = all.equal, c(-180, 180), angle) == TRUE)){**

**hjust <- 0**

**vjust <- 0.5**

**}else if((angle > -360 & angle < -270) | (angle > 0 & angle < 90)){**

**hjust <- 1**

**vjust <- 0**

**}else if((angle > -270 & angle < -180) | (angle > 90 & angle < 180)){**

**hjust <- 0**

**vjust <- 0**

**}else if((angle > -180 & angle < -90) | (angle > 180 & angle < 270)){**

**hjust <- 0**

**vjust <- 1**

**}else if((angle > -90 & angle < 0) | (angle > 270 & angle < 360)){**

**hjust <- 1**

**vjust <- 1**

**}**

**}**

*# end justifications*

**output <- list(angle = angle, hjust = hjust, vjust = vjust)**

**return(output)**

**}**

######## fun\_gg\_get\_legend() #### get the legend of ggplot objects



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

**fun\_gg\_get\_legend <- function(ggplot\_built, fun.name = NULL, lib.path = NULL){**

*# AIM*

*# get legend of ggplot objects*

*# # from https://stackoverflow.com/questions/12539348/ggplot-separate-legend-and-plot*

*# ARGUMENTS*

*# ggplot\_built: a ggplot build object*

*# fun.name: single character string indicating the name of the function using fun\_gg\_get\_legend(). Ignored if NULL*

*# lib.path: character vector specifying the absolute pathways of the directories containing the required packages if not in the default directories. Ignored if NULL*

*# REQUIRED PACKAGES*

*# ggplot2*

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

*# fun\_check()*

*# fun\_pack()*

*# RETURN*

*# a list of class c("gtable", "gTree", "grob", "gDesc"), providing legend information of ggplot\_built objet, or NULL if the ggplot\_built object has no legend*

*# EXAMPLES*

*# Simple example*

*# obs1 <- data.frame(time = 1:20, group = rep(c("CLASS\_1", "CLASS\_2"), times = 10)) ; p <- ggplot2::ggplot() + ggplot2::geom\_point(data = obs1, mapping = ggplot2::aes(x = group, y = time, fill = group)) ; fun\_gg\_get\_legend(ggplot\_built = ggplot2::ggplot\_build(p))*

*# Error message because no legend in the ggplot*

*# obs1 <- data.frame(time = 1:20, group = rep(c("CLASS\_1", "CLASS\_2"), times = 10)) ; p <- ggplot2::ggplot() + ggplot2::geom\_point(data = obs1, mapping = ggplot2::aes(x = group, y = time)) ; fun\_gg\_get\_legend(ggplot\_built = ggplot2::ggplot\_build(p))*

*# DEBUGGING*

*# obs1 <- data.frame(time = 1:20, group = rep(c("CLASS\_1", "CLASS\_2"), times = 10)) ; p <- ggplot2::ggplot() + ggplot2::geom\_point(data = obs1, mapping = ggplot2::aes(x = group, y = time)) ; ggplot\_built = ggplot2::ggplot\_build(p) ; fun.name = NULL ; lib.path = NULL*

*# function name*

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

*# end function name*

*# required function checking*

**req.function <- c(**

**"fun\_check",**

**"fun\_pack"**

**)**

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

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

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

**stop(paste0("\n\n================\n\n", tempo.cat, "\n\n================\n\n"), call. = FALSE)** *# == in stop() to be able to add several messages between ==*

**}**

**}**

*# end required function checking*

*# argument primary checking*

*# arg with no default values*

**if(any(missing(ggplot\_built))){**

**tempo.cat <- paste0("ERROR IN ", function.name, ": ARGUMENTS ggplot\_built HAVE NO DEFAULT VALUE AND REQUIRE ONE")**

**stop(paste0("\n\n================\n\n", tempo.cat, "\n\n================\n\n"), call. = FALSE)** *# == in stop() to be able to add several messages between ==*

**}**

*# end arg with no default values*

*# using fun\_check()*

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

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

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

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

**tempo <- fun\_check(data = ggplot\_built, class = "ggplot\_built", mode = "list", fun.name = function.name) ; eval(ee)**

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

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

**}**

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

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

**}**

**if( ! is.null(arg.check)){**

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

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

**}**

**}**

*# end using fun\_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\_check\_dev)) # activate this line and use the function (with no arguments left as NULL) to check arguments status and if they have been checked using fun\_check()*

*# end argument primary checking*

*# second round of checking*

*# dealing with NA*

**if(any(is.na(ggplot\_built)) | any(is.na(fun.name)) | any(is.na(lib.path))){**

**tempo.cat <- paste0("ERROR IN ", function.name, ": NO ARGUMENT CAN HAVE NA VALUES")**

**stop(paste0("\n\n================\n\n", tempo.cat, "\n\n================\n\n"), call. = FALSE)** *# == in stop() to be able to add several messages between ==*

**}**

*# end dealing with NA*

*# dealing with NULL*

**if(is.null(ggplot\_built)){**

**tempo.cat <- paste0("ERROR IN ", function.name, "\nggplot\_built ARGUMENT CANNOT BE NULL")**

**stop(paste0("\n\n================\n\n", tempo.cat, "\n\n================\n\n"), call. = FALSE)** *# == in stop() to be able to add several messages between ==*

**}**

*# end dealing with NULL*

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

**if( ! all(dir.exists(lib.path))){**

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

**stop(paste0("\n\n================\n\n", tempo.cat, "\n\n================\n\n"), call. = FALSE)** *# == in stop() to be able to add several messages between ==*

**}**

**}**

*# end second round of checking*

*# package checking*

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

*# end package checking*

*# main code*

**win.nb <- dev.cur()**

**pdf(file = NULL)**

**tmp <- ggplot2::ggplot\_gtable(ggplot\_built)**

*# BEWARE with ggplot\_gtable : open a blanck device https://stackoverflow.com/questions/17012518/why-does-this-r-ggplot2-code-bring-up-a-blank-display-device*

**invisible(dev.off())**

**if(win.nb > 1){** *# to go back to the previous active device, if == 1 means no opened device*

**dev.set(win.nb)**

**}**

**leg <- which(sapply(tmp$grobs, function(x) x$name) == "guide-box")**

**if(length(leg) == 0){**

**legend <- NULL**

**}else{**

**legend <- tmp$grobs[[leg]]**

**}**

**return(legend)**

**}**

######## fun\_gg\_point\_rast() #### ggplot2 raster scatterplot layer



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

**fun\_gg\_point\_rast <- function(data = NULL, mapping = NULL, stat = "identity", position = "identity", ..., na.rm = FALSE, show.legend = NA, inherit.aes = TRUE, raster.width = NULL, raster.height = NULL, raster.dpi = 300, inactivate = TRUE, lib.path = NULL){**

*# AIM*

*# equivalent to ggplot2::geom\_point() but in raster mode*

*# use it like ggplot2::geom\_point() with the main raster.dpi additional argument*

*# WARNINGS*

*# can be long to generate the plot*

*# use a square plot region. Otherwise, the dots will have ellipsoid shape*

*# solve the transparency problems with some GUI*

*# this function is derived from the geom\_point\_rast() function, created by Viktor Petukhov , and present in the ggrastr package (https://rdrr.io/github/VPetukhov/ggrastr/src/R/geom-point-rast.R, MIT License, Copyright (c) 2017 Viktor Petukhov). Has been placed here to minimize package dependencies*

*# ARGUMENTS*

*# classical arguments of geom\_point(), shown here https://rdrr.io/github/VPetukhov/ggrastr/man/geom\_point\_rast.html*

*# raster.width : width of the result image (in inches). Default: deterined by the current device parameters*

*# raster.height: height of the result image (in inches). Default: deterined by the current device parameters*

*# raster.dpi: resolution of the result image*

*# inactivate: logical. Inactivate the fun.name argument of the fun\_check() function? If TRUE, the name of the fun\_check() function in error messages coming from this function. Use TRUE if fun\_gg\_point\_rast() is used like this: eval(parse(text = "fun\_gg\_point\_rast"))*

*# lib.path: character vector specifying the absolute pathways of the directories containing the required packages if not in the default directories. Ignored if NULL*

*# REQUIRED PACKAGES*

*# ggplot2*

*# grid*

*# Cairo*

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

*# fun\_check()*

*# fun\_pack()*

*# RETURN*

*# a raster scatter plot*

*# EXAMPLES*

*# Two pdf in the current directory*

*# set.seed(1) ; data1 = data.frame(x = rnorm(100000), y = rnorm(10000)) ; fun\_open(pdf.name = "Raster") ; ggplot2::ggplot() + fun\_gg\_point\_rast(data = data1, mapping = ggplot2::aes(x = x, y = y)) ; fun\_open(pdf.name = "Vectorial") ; ggplot2::ggplot() + ggplot2::geom\_point(data = data1, mapping = ggplot2::aes(x = x, y = y)) ; dev.off() ; dev.off()*

*# DEBUGGING*

*#*

*# function name*

**if(all(inactivate == FALSE)){** *# inactivate has to be used here but will be fully checked below*

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

**}else if(all(inactivate == TRUE)){**

**function.name <- NULL**

**}else{**

**tempo.cat <- paste0("ERROR IN fun\_gg\_point\_rast(): CODE INCONSISTENCY 1")**

**stop(paste0("\n\n================\n\n", tempo.cat, "\n\n================\n\n"), call. = FALSE)** *# == in stop() to be able to add several messages between ==*

**}**

*# end function name*

*# required function checking*

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

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

**stop(paste0("\n\n================\n\n", tempo.cat, "\n\n================\n\n"), call. = FALSE)** *# == in stop() to be able to add several messages between ==*

**}**

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

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

**stop(paste0("\n\n================\n\n", tempo.cat, "\n\n================\n\n"), call. = FALSE)** *# == in stop() to be able to add several messages between ==*

**}**

*# end required function checking*

*# argument checking*

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

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

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

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

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

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

**}**

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

**tempo <- fun\_check(data = mapping, class = "uneval", typeof = "list", fun.name = function.name) ; eval(ee)** *# aes() is tested*

**}**

*# stat and position not tested because too complicate*

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

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

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

**if( ! is.null(raster.width)){**

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

**}**

**if( ! is.null(raster.height)){**

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

**}**

**tempo <- fun\_check(data = raster.dpi, class = "integer", length = 1, double.as.integer.allowed = TRUE, neg.values = FALSE, fun.name = function.name) ; eval(ee)**

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

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

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

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

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

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

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

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

**}**

**}**

**}**

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

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

**}**

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

*# end argument checking*

*# package checking*

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

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

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

*# end package checking*

*# additional functions*

**DrawGeomPointRast <- function(data, panel\_params, coord, na.rm = FALSE, raster.width = NULL, raster.height= NULL, raster.dpi = 300){**

**if (is.null(raster.width)){**

**raster.width <- par('fin')[1]**

**}**

**if (is.null(raster.height)){**

**raster.height <- par('fin')[2]**

**}**

**prev\_dev\_id <- dev.cur()**

**p <- ggplot2::GeomPoint$draw\_panel(data, panel\_params, coord)**

**dev\_id <- Cairo::Cairo(type='raster', width = raster.width\*raster.dpi, height = raster.height\*raster.dpi, dpi = raster.dpi, units = 'px', bg = "transparent")[1]**

**grid::pushViewport(grid::viewport(width = 1, height = 1))**

**grid::grid.points(x = p$x, y = p$y, pch = p$pch, size = p$size,**

**name = p$name, gp = p$gp, vp = p$vp, draw = T)**

**grid::popViewport()**

**cap <- grid::grid.cap()**

**invisible(dev.off(dev\_id))**

**invisible(dev.set(prev\_dev\_id))**

**grid::rasterGrob(cap, x = 0, y = 0, width = 1, height = 1, default.units = "native", just = c("left","bottom"))**

**}**

*# end additional functions*

*# main code*

**GeomPointRast <- ggplot2::ggproto("GeomPointRast", ggplot2::GeomPoint, draw\_panel = DrawGeomPointRast)**

**ggplot2::layer(**

**data = data,**

**mapping = mapping,**

**stat = stat,**

**geom = GeomPointRast,**

**position = position,**

**show.legend = show.legend,**

**inherit.aes = inherit.aes,**

**params = list(**

**na.rm = na.rm,**

**raster.width = raster.width,**

**raster.height = raster.height,**

**raster.dpi = raster.dpi,**

**...**

**)**

**)**

*# end main code*

**}**

######## fun\_gg\_boxplot() #### ggplot2 boxplot + background dots if required

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

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

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

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

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

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

*# AIM*

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

*# see also:*

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

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

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

*# ARGUMENTS*

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

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

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

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

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

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

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

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

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

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

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

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

*# title: character string of the graph title*

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

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

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

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

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

*# add: character string allowing to add more ggplot2 features (dots, lines, themes, etc.). BEWARE: (1) must start with "+" just after the simple or double opening quote (no space, end of line, carriage return, etc., allowed), (2) must finish with ")" just before the simple or double closing quote (no space, end of line, carriage return, etc., allowed) and (3) each function must be preceded by "ggplot2::" (for instance: "ggplot2::coord\_flip()). If the character string contains the "ggplot2::theme" string, then internal ggplot2 theme() and theme\_classic() functions will be inactivated to be reused by add. BEWARE: handle this argument with caution since added functions can create conflicts with the preexisting internal ggplot2 functions*

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

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

*# REQUIRED PACKAGES*

*# ggplot2*

*# reshape2*

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

*# fun\_check()*

*# fun\_pack()*

*# fun\_round()*

*# RETURN*

*# a heatmap if plot argument is TRUE*

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

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

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

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

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

*# EXAMPLES*

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

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

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

*# diagonal matrix*

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

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

*# error message*

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

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

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

*#### NICE REPRESENTATION*

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

*#### SINGLE GEOMETRIC LAYER*

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

*# legend name*

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

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

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

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

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

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

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

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

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

*#### MULTI GEOMETRIC LAYERS*

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

*# single layer (as examples above)*

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

*# simple example of two layers*

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

*# color of dots. Example (1)*

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

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

*# set.seed(1) ; obs1 <- data.frame(km = rnorm(1000, 22, 3), time = rnorm(1000, 22, 3), group1 = "GROUP1") ; obs2 <-data.frame(km = rnorm(1000, 30, 3), time = rnorm(1000, 30, 3), group2 = "GROUP2") ; set.seed(NULL) ; fun\_gg\_scatter(data1 = list(L1 = obs1, L2 = obs2), x = list(L1 = "km", L2 = "km"), y = list(L1 = "time", L2 = "time"), categ = list(L1 = "group1", L2 = "group2"), legend.name = list(L1 = NULL, L2 = NULL), geom = list(L1 = "geom\_point", L2 = "geom\_point"), alpha = list(L1 = 0.5, L2 = 0.5), color = list(L1 = "coral", L2 = "green"))*

*# color of dots. Example (3) when dots are in different categories (default colors)*

*# set.seed(1) ; obs1 <- data.frame(km = rnorm(1000, 22, 3), time = rnorm(1000, 22, 3), group1 = rep(c("A1", "A2"), each = 500)) ; obs2 <-data.frame(km = rnorm(1000, 30, 3), time = rnorm(1000, 30, 3), group2 = rep(c("G1", "G2"), each = 500)) ; set.seed(NULL) ; fun\_gg\_scatter(data1 = list(L1 = obs1, L2 = obs2), x = list(L1 = "km", L2 = "km"), y = list(L1 = "time", L2 = "time"), , categ = list(L1 = "group1", L2 = "group2"), geom = list(L1 = "geom\_point", L2 = "geom\_point"), alpha = list(L1 = 0.5, L2 = 0.5))*

*# color of dots. Example (3) when dots are in different categories. A single color mentionned per layer is applied to all the dots of the layer*

*# set.seed(1) ; obs1 <- data.frame(km = rnorm(1000, 22, 3), time = rnorm(1000, 22, 3), group1 = rep(c("A1", "A2"), each = 500)) ; obs2 <-data.frame(km = rnorm(1000, 30, 3), time = rnorm(1000, 30, 3), group2 = rep(c("G1", "G2"), each = 500)) ; set.seed(NULL) ; fun\_gg\_scatter(data1 = list(L1 = obs1, L2 = obs2), x = list(L1 = "km", L2 = "km"), y = list(L1 = "time", L2 = "time"), , categ = list(L1 = "group1", L2 = "group2"), geom = list(L1 = "geom\_point", L2 = "geom\_point"), alpha = list(L1 = 0.5, L2 = 0.5), color = list(L1 = "coral", L2 = "green"))*

*# color of dots. Example (5) when dots are in different categories, with one color per category in each layer*

*# set.seed(1) ; obs1 <- data.frame(km = rnorm(1000, 22, 3), time = rnorm(1000, 22, 3), group1 = rep(c("A1", "A2"), each = 500)) ; obs2 <-data.frame(km = rnorm(1000, 30, 3), time = rnorm(1000, 30, 3), group2 = rep(c("G1", "G2"), each = 500)) ; set.seed(NULL) ; fun\_gg\_scatter(data1 = list(L1 = obs1, L2 = obs2), x = list(L1 = "km", L2 = "km"), y = list(L1 = "time", L2 = "time"), , categ = list(L1 = "group1", L2 = "group2"), geom = list(L1 = "geom\_point", L2 = "geom\_point"), alpha = list(L1 = 0.5, L2 = 0.5), color = list(L1 = c("coral", "blue"), L2 = c("green", "black")))*

*# color of dots. Example (4) when dots are in different categories. Numbers can be used if ggplot colors are desired*

*# set.seed(1) ; obs1 <- data.frame(km = rnorm(1000, 22, 3), time = rnorm(1000, 22, 3), group1 = rep(c("A1", "A2"), each = 500)) ; obs2 <-data.frame(km = rnorm(1000, 30, 3), time = rnorm(1000, 30, 3), group2 = rep(c("G1", "G2"), each = 500)) ; set.seed(NULL) ; fun\_gg\_scatter(data1 = list(L1 = obs1, L2 = obs2), x = list(L1 = "km", L2 = "km"), y = list(L1 = "time", L2 = "time"), , categ = list(L1 = "group1", L2 = "group2"), geom = list(L1 = "geom\_point", L2 = "geom\_point"), alpha = list(L1 = 0.5, L2 = 0.5), color = list(L1 = 1:2, L2 = c(4, 7)))*

*# color of dots. Example (7) when dots are in different categories, with colors as a data frame column. BEWARE: one color per category must be respected (try also numbers). BEWARE: in color argument, if the column of the data frame does not exist, color can be still displayed (L2 = obs2$notgood is equivalent to L2 = NULL). Such situation is reported in the warning messages (see below)*

*# set.seed(1) ; obs1 <- data.frame(km = rnorm(1000, 22, 3), time = rnorm(1000, 22, 3), group1 = rep(c("A1", "A2"), each = 500), col1 = rep(c("coral", "blue"), each = 500)) ; obs2 <-data.frame(km = rnorm(1000, 30, 3), time = rnorm(1000, 30, 3), group2 = rep(c("G1", "G2"), each = 500), col2 = rep(c("green", "black"), each = 500)) ; set.seed(NULL) ; fun\_gg\_scatter(data1 = list(L1 = obs1, L2 = obs2), x = list(L1 = "km", L2 = "km"), y = list(L1 = "time", L2 = "time"), , categ = list(L1 = "group1", L2 = "group2"), geom = list(L1 = "geom\_point", L2 = "geom\_point"), alpha = list(L1 = 0.5, L2 = 0.5), color = list(L1 = obs1$col1, L2 = obs2$col2))*

*# color of dots. Example (8) when dots are in different categories, with colors as a data frame column. Easiest way is not recommended with mutiple layers*

*# set.seed(1) ; obs1 <- data.frame(km = rnorm(1000, 22, 3), time = rnorm(1000, 22, 3), group1 = rep(c("A1", "A2"), each = 500), col1 = rep(c("coral", "blue"), each = 500)) ; obs2 <-data.frame(km = rnorm(1000, 30, 3), time = rnorm(1000, 30, 3), group2 = rep(c("G1", "G2"), each = 500), col2 = rep(c("green", "black"), each = 500)) ; set.seed(NULL) ; fun\_gg\_scatter(data1 = list(L1 = obs1, L2 = obs2), x = list(L1 = "km", L2 = "km"), y = list(L1 = "time", L2 = "time"), , categ = list(L1 = "group1", L2 = "group2"), geom = list(L1 = "geom\_point", L2 = "geom\_point"), alpha = list(L1 = 0.5, L2 = 0.5), color = list(L1 = as.numeric(obs1$group1), L2 = as.numeric(obs2$group2)))*

*# legend name*

*# set.seed(1) ; obs1 <- data.frame(km = rnorm(1000, 22, 3), time = rnorm(1000, 22, 3), group1 = rep(c("A1", "A2"), each = 500)) ; obs2 <-data.frame(km = rnorm(1000, 30, 3), time = rnorm(1000, 30, 3), group2 = rep(c("G1", "G2"), each = 500)) ; set.seed(NULL) ; fun\_gg\_scatter(data1 = list(L1 = obs1, L2 = obs2), x = list(L1 = "km", L2 = "km"), y = list(L1 = "time", L2 = "time"), , categ = list(L1 = "group1", L2 = "group2"), legend.name = list(L1 = "CLASS A", L2 = "CLASS G"), geom = list(L1 = "geom\_point", L2 = "geom\_point"), alpha = list(L1 = 0.5, L2 = 0.5))*

*# different geom features. Example (1) with 5 layers. Note that order in data1 defines the overlay order (from below to above) and the order in the legend (from top to bottom)*

*# set.seed(1) ; obs1 <- data.frame(km = rnorm(1000, 22, 3), time = rnorm(1000, 22, 3), group1 = rep(c("A1", "A2"), each = 500)) ; obs2 <-data.frame(km = rnorm(1000, 30, 3), time = rnorm(1000, 30, 3), group2 = rep(c("G1", "G2"), each = 500)) ; set.seed(NULL) ; obs3 <- data.frame(time = c(29, 31), group3 = c("HORIZ.THRESHOLD.1", "HORIZ.THRESHOLD.2")) ; obs4 <- data.frame(km = 26, group4 = "VERTIC.THRESHOLD") ; obs5 <- data.frame(km = seq(1, 100, 0.1), time = 7\*seq(1, 100, 0.1)^0.5, group5 = "FUNCTION") ; fun\_gg\_scatter(data1 = list(L1 = obs1, L2 = obs2, L3 = obs3, L4 = obs4, L5 = obs5), x = list(L1 = "km", L2 = "km", L3 = NULL, L4 = "km", L5 = "km"), y = list(L1 = "time", L2 = "time", L3 = "time", L4 = NULL, L5 = "time"), categ = list(L1 = "group1", L2 = "group2", L3 = "group3", L4 = "group4", L5 = "group5"), geom = list(L1 = "geom\_point", L2 = "geom\_point", L3 = "geom\_hline", L4 = "geom\_vline", L5 = "geom\_line"), alpha = list(L1 = 0.5, L2 = 0.5, L3 = 0.5, L4 = 0.5, L5 = 0.5), xlim = c(10, 40), ylim = c(10, 40), classic = TRUE, line.size = 0.75)*

*# layer transparency. One transparency defined by layer (from 0 invisible to 1 opaque). Note that for lines, transparency in not applied in the legend to prevent a ggplot2 bug (https://github.com/tidyverse/ggplot2/issues/2452)*

*# set.seed(1) ; obs1 <- data.frame(km = rnorm(1000, 22, 3), time = rnorm(1000, 22, 3), group1 = rep(c("A1", "A2"), each = 500)) ; obs2 <-data.frame(km = rnorm(1000, 30, 3), time = rnorm(1000, 30, 3), group2 = rep(c("G1", "G2"), each = 500)) ; set.seed(NULL) ; fun\_gg\_scatter(data1 = list(L1 = obs1, L2 = obs2), x = list(L1 = "km", L2 = "km"), y = list(L1 = "time", L2 = "time"), , categ = list(L1 = "group1", L2 = "group2"), geom = list(L1 = "geom\_point", L2 = "geom\_point"), alpha = list(L1 = 1, L2 = 0.1))*

*# other different example of mutiple geom features are shown in the fun\_segmentation function*

*#### OTHER GRAPHIC ARGUMENTS*

*# dot size (line.size argument controls size of lines)*

*# obs1 <- data.frame(km = 2:7, time = (2:7)^2, group = c("A", "A", "A", "B", "B", "B")) ; obs1 ; fun\_gg\_scatter(data1 = obs1, x = "km", y = "time", dot.size = 5)*

*# axis management: examples are shown for x-axis but are identical for y-axis*

*# x-axis limits. Example (1)*

*# obs1 <- data.frame(km = 2:7, time = (2:7)^2, group = c("A", "A", "A", "B", "B", "B")) ; obs1 ; fun\_gg\_scatter(data1 = obs1, x = "km", y = "time", xlim = c(-1, 25))*

*# x-axis limits. Example (2) showing that order matters in ylim argument*

*# obs1 <- data.frame(km = 2:7, time = (2:7)^2, group = c("A", "A", "A", "B", "B", "B")) ; obs1 ; fun\_gg\_scatter(data1 = obs1, x = "km", y = "time", xlim = c(25, -1))*

*# log scale. Example (1). BEWARE: x column must be log, otherwise incoherent scale (see below warning message with the return argument)*

*# obs1 <- data.frame(km = 2:7, time = (2:7)^2, group = c("A", "A", "A", "B", "B", "B")) ; obs1 ; fun\_gg\_scatter(data1 = obs1, x = "km", y = "time", xlog = "log10")*

*# log scale. Example (2). BEWARE: values of the xlim must be in the corresponding log*

*# obs1 <- data.frame(km = 2:7, time = (2:7)^2, group = c("A", "A", "A", "B", "B", "B")) ; obs1 ; fun\_gg\_scatter(data1 = obs1, x = "km", y = "time", xlog = "log10", xlim = c(1, 10))*

*# tick number. Example (1). Note that the final number shown is approximate*

*# obs1 <- data.frame(km = 2:7, time = (2:7)^2, group = c("A", "A", "A", "B", "B", "B")) ; obs1 ; fun\_gg\_scatter(data1 = obs1, x = "km", y = "time", x.tick.nb = 6)*

*# tick number. Example (2) using a log2 scale*

*# obs1 <- data.frame(km = 2:7, time = (2:7)^2, group = c("A", "A", "A", "B", "B", "B")) ; obs1 ; fun\_gg\_scatter(data1 = obs1, x = "km", y = "time", xlog = "log2", x.tick.nb = 6)*

*# tick number. Example (3) using a log10 scale*

*# obs1 <- data.frame(km = 2:7, time = (2:7)^2, group = c("A", "A", "A", "B", "B", "B")) ; obs1 ; fun\_gg\_scatter(data1 = obs1, x = "km", y = "time", xlog = "log10", x.tick.nb = 6)*

*# tick number. Example (4) using a log10 scale: the reverse x-axis correctly deal with log10 scale*

*# obs1 <- data.frame(km = 2:7, time = (2:7)^2, group = c("A", "A", "A", "B", "B", "B")) ; obs1 ; fun\_gg\_scatter(data1 = obs1, x = "km", y = "time", xlog = "log10", xlim = c(7, 2))*

*# secondary tick number. Example (1)*

*# obs1 <- data.frame(km = 2:7, time = (2:7)^2, group = c("A", "A", "A", "B", "B", "B")) ; obs1 ; fun\_gg\_scatter(data1 = obs1, x = "km", y = "time", x.inter.tick.nb = 4)*

*# secondary ticks. Example (2) not for log2 and log10 scales (see below warning message with the return argument)*

*# obs1 <- data.frame(km = 2:7, time = (2:7)^2, group = c("A", "A", "A", "B", "B", "B")) ; obs1 ; fun\_gg\_scatter(data1 = obs1, x = "km", y = "time", xlog = "log10", x.inter.tick.nb = 4)*

*# extra margins. To avoid dot cuts*

*# obs1 <- data.frame(km = 2:7, time = (2:7)^2, group = c("A", "A", "A", "B", "B", "B")) ; obs1 ; fun\_gg\_scatter(data1 = obs1, x = "km", y = "time", x.left.extra.margin = 0.25, x.right.extra.margin = 0.25)*

*# include zero in both the x-axis and y-xis*

*# obs1 <- data.frame(km = 2:7, time = (2:7)^2, group = c("A", "A", "A", "B", "B", "B")) ; obs1 ; fun\_gg\_scatter(data1 = obs1, x = "km", y = "time", xy.include.zero = TRUE)*

*# graph title, text size and legend display*

*# obs1 <- data.frame(km = 2:7, time = (2:7)^2, group = c("A", "A", "A", "B", "B", "B")) ; obs1 ; fun\_gg\_scatter(data1 = obs1, x = "km", y = "time", categ = "group", text.size = 8, title = "GRAPH1", title.text.size = 16, show.legend = TRUE)*

*# raster display. This switchs from vectorial mode to raster mode. The display can takes some time, but this is easier to export and handle than vectorial display*

*# set.seed(1) ; obs1 <- data.frame(km = rnorm(100000, 22, 3), time = rnorm(100000, 22, 3)) ; set.seed(NULL) ; fun\_gg\_scatter(data1 = obs1, x = "km", y = "time", raster = TRUE)*

*# classic representation (use grid = TRUE to display the background lines of the y axis ticks)*

*# obs1 <- data.frame(km = 2:7, time = (2:7)^2, group = c("A", "A", "A", "B", "B", "B")) ; obs1 ; fun\_gg\_scatter(data1 = obs1, x = "km", y = "time", classic = TRUE, grid = FALSE)*

*# graphic info. Example (1)*

*# obs1 <- data.frame(km = 2:7, time = (2:7)^2, group = c("A", "A", "A", "B", "B", "B")) ; fun\_gg\_scatter(data1 = obs1, x = "km", y = "time", return = TRUE)*

*# graphic info. Example (2) of assignation and warning message display*

*# obs1 <- data.frame(km = 2:7, time = (2:7)^2, group = c("A", "A", "A", "B", "B", "B")) ; output <- fun\_gg\_scatter(data1 = obs1, x = "km", y = "time", xlog = "log10", return = TRUE) ; cat(output$warn)*

*# add ggplot2 functions*

*# obs1 <- data.frame(km = 2:7, time = (2:7)^2, group = c("A", "A", "A", "B", "B", "B")) ; obs1 ; fun\_gg\_scatter(data1 = obs1, x = "km", y = "time", add = "+ggplot2::theme\_classic()")*

*# all the arguments*

*# set.seed(1) ; obs1 <- data.frame(km = rnorm(1000, 10, 3), time = rnorm(1000, 10, 3), group1 = rep(c("A1", "A2"), 500)) ; obs2 <-data.frame(km = rnorm(1000, 15, 3), time = rnorm(1000, 15, 3), group2 = rep(c("G1", "G2"), 500)) ; set.seed(NULL) ; obs1$L1$km[2:3] <- NA ; fun\_gg\_scatter(data1 = list(L1 = obs1, L2 = obs2), x = list(L1 = "km", L2 = "km"), y = list(L1 = "time", L2 = "time"), categ = list(L1 = "group1", L2 = "group2"), legend.name = NULL, color = list(L1 = 4:5, L2 = 7:8), geom = list(L1 = "geom\_point", L2 = "geom\_point"), alpha = list(L1 = 0.5, L2 = 0.5), dot.size = 3, line.size = 0.5, xlim = c(1, 25), xlab = "KM", xlog = "no", x.tick.nb = 10, x.inter.tick.nb = 1, x.left.extra.margin = 0, x.right.extra.margin = 0, ylim = c(1, 25), ylab = "TIME (s)", ylog = "log10", y.tick.nb = 5, y.inter.tick.nb = NULL, y.top.extra.margin = 0, y.bottom.extra.margin = 0, xy.include.zero = TRUE, text.size = 12, title = "", title.text.size = 8, show.legend = TRUE, classic = FALSE, grid = FALSE, raster = FALSE, vectorial.limit = NULL, return = FALSE, plot = TRUE, add = NULL, warn.print = TRUE, lib.path = NULL)*

*# DEBUGGING*

*# data1 = matrix(1:16, ncol = 4) ; legend.name1 = "" ; low.color1 = "blue" ; mid.color1 = "white" ; high.color1 = "red" ; limit1 = NULL ; midpoint1 = NULL ; data2 = matrix(rep(c(1,0,0,0), 4), ncol = 4) ; color2 = "black" ; alpha2 = 0.5 ; invert2 = FALSE ; text.size = 12 ; title = "" ; title.text.size = 12 ; show.scale = TRUE ; rotate = FALSE ; return = FALSE ; plot = TRUE ; add = NULL ; warn.print = TRUE ; lib.path = NULL*

*# function name*

**function.name <- paste0(as.list(match.call(expand.dots=FALSE))[[1]], "()")**

*# end function name*

*# required function checking*

**if(length(utils::find("fun\_check", mode = "function")) == 0){**

**tempo.cat <- paste0("ERROR IN ", function.name, ": REQUIRED fun\_check() FUNCTION IS MISSING IN THE R ENVIRONMENT")**

**stop(paste0("\n\n================\n\n", tempo.cat, "\n\n================\n\n"), call. = FALSE)** *# == in stop() to be able to add several messages between ==*

**}**

**if(length(utils::find("fun\_pack", mode = "function")) == 0){**

**tempo.cat <- paste0("ERROR IN ", function.name, ": REQUIRED fun\_pack() FUNCTION IS MISSING IN THE R ENVIRONMENT")**

**stop(paste0("\n\n================\n\n", tempo.cat, "\n\n================\n\n"), call. = FALSE)** *# == in stop() to be able to add several messages between ==*

**}**

**if(length(utils::find("fun\_round", mode = "function")) == 0){**

**tempo.cat <- paste0("ERROR IN ", function.name, ": REQUIRED fun\_round() FUNCTION IS MISSING IN THE R ENVIRONMENT")**

**stop(paste0("\n\n================\n\n", tempo.cat, "\n\n================\n\n"), call. = FALSE)** *# == in stop() to be able to add several messages between ==*

**}**

*# end required function checking*

*# no reserved words required for this function*

*# argument checking*

**arg.check <- NULL** *#*

**text.check <- NULL** *#*

**checked.arg.names <- NULL** *# for function debbuging: used by r\_debugging\_tools*

**ee <- expression(arg.check <- c(arg.check, tempo$problem) , text.check <- c(text.check, tempo$text) , checked.arg.names <- c(checked.arg.names, tempo$fun.name))**

**if(all(is.matrix(data1))){**

**tempo <- fun\_check(data = data1, class = "matrix", mode = "numeric", na.contain = TRUE, fun.name = function.name) ; eval(ee)**

**}else if(all(is.data.frame(data1))){**

**tempo <- fun\_check(data = data1, class = "data.frame", length = 3, fun.name = function.name) ; eval(ee)**

**if(tempo$problem == FALSE){**

*# structure of reshape2::melt() data frame*

**tempo <- fun\_check(data = data1[, 1], data.name = "COLUMN 1 OF data1 (reshape2::melt() DATA FRAME)", typeof = "integer", fun.name = function.name) ; eval(ee)**

**tempo <- fun\_check(data = data1[, 2], data.name = "COLUMN 2 OF data1 (reshape2::melt() DATA FRAME)", typeof = "integer", fun.name = function.name) ; eval(ee)**

**tempo <- fun\_check(data = data1[, 3], data.name = "COLUMN 3 OF data1 (reshape2::melt() DATA FRAME)", mode = "numeric", na.contain = TRUE, fun.name = function.name) ; eval(ee)**

**}**

**}else{**

**tempo.cat <- paste0("ERROR IN ", function.name, ": THE data1 ARGUMENT MUST BE A NUMERIC MATRIX OR A DATA FRAME OUTPUT OF THE reshape::melt() FUNCTION")**

**text.check <- c(text.check, tempo.cat)**

**arg.check <- c(arg.check, TRUE)**

**}**

**tempo <- fun\_check(data = legend.name1, class = "character", length = 1, fun.name = function.name) ; eval(ee)**

**tempo <- fun\_check(data = low.color1, class = "character", length = 1, fun.name = function.name) ; eval(ee)**

**if(tempo$problem == FALSE & ! (all(low.color1 %in% colors() | grepl(pattern = "^#", low.color1)))){** *# check that all strings of low.color1 start by #*

**tempo.cat <- paste0("ERROR IN ", function.name, ": low.color1 ARGUMENT MUST BE A HEXADECIMAL COLOR VECTOR STARTING BY # AND/OR COLOR NAMES GIVEN BY colors()")**

**text.check <- c(text.check, tempo.cat)**

**arg.check <- c(arg.check, TRUE)**

**}**

**if( ! is.null(mid.color1)){**

**tempo <- fun\_check(data = mid.color1, class = "character", length = 1, fun.name = function.name) ; eval(ee)**

**if(tempo$problem == FALSE & ! (all(mid.color1 %in% colors() | grepl(pattern = "^#", mid.color1)))){** *# check that all strings of mid.color1 start by #*

**tempo.cat <- paste0("ERROR IN ", function.name, ": mid.color1 ARGUMENT MUST BE A HEXADECIMAL COLOR VECTOR STARTING BY # AND/OR COLOR NAMES GIVEN BY colors()")**

**text.check <- c(text.check, tempo.cat)**

**arg.check <- c(arg.check, TRUE)**

**}**

**}**

**tempo <- fun\_check(data = high.color1, class = "character", length = 1, fun.name = function.name) ; eval(ee)**

**if(tempo$problem == FALSE & ! (all(high.color1 %in% colors() | grepl(pattern = "^#", high.color1)))){** *# check that all strings of high.color1 start by #*

**tempo.cat <- paste0("ERROR IN ", function.name, ": high.color1 ARGUMENT MUST BE A HEXADECIMAL COLOR VECTOR STARTING BY # AND/OR COLOR NAMES GIVEN BY colors()")**

**text.check <- c(text.check, tempo.cat)**

**arg.check <- c(arg.check, TRUE)**

**}**

**if( ! is.null(limit1)){**

**tempo <- fun\_check(data = limit1, class = "vector", mode = "numeric", length = 2, fun.name = function.name) ; eval(ee)**

**if(tempo$problem == FALSE & any(limit1 %in% c(Inf, -Inf))){**

**tempo.cat <- paste0("ERROR IN ", function.name, ": limit1 ARGUMENT CANNOT CONTAIN -Inf OR Inf VALUES")**

**text.check <- c(text.check, tempo.cat)**

**arg.check <- c(arg.check, TRUE)**

**}**

**}**

**if( ! is.null(midpoint1)){**

**tempo <- fun\_check(data = midpoint1, class = "vector", mode = "numeric", length = 1, fun.name = function.name) ; eval(ee)**

**}**

**if( ! is.null(data2)){**

**if(all(is.matrix(data2))){**

**tempo <- fun\_check(data = data2, class = "matrix", mode = "numeric", fun.name = function.name) ; eval(ee)**

**if(tempo$problem == FALSE & ! all(unique(data2) %in% c(0,1))){**

**tempo.cat <- paste0("ERROR IN ", function.name, ": MATRIX IN data2 MUST BE MADE OF 0 AND 1 ONLY (MASK MATRIX)")**

**text.check <- c(text.check, tempo.cat)**

**arg.check <- c(arg.check, TRUE)**

**}else if(tempo$problem == FALSE & all(is.matrix(data1)) & ! identical(dim(data1), dim(data2))){** *# matrix and matrix*

**tempo.cat <- paste0("ERROR IN ", function.name, ": MATRIX DIMENSION IN data2 MUST BE IDENTICAL AS MATRIX DIMENSION IN data1. HERE IT IS RESPECTIVELY:\n", paste(dim(data2), collapse = " "), "\n", paste(dim(data1), collapse = " "))**

**text.check <- c(text.check, tempo.cat)**

**arg.check <- c(arg.check, TRUE)**

**}else if(tempo$problem == FALSE & all(is.data.frame(data1)) & nrow(data1) != prod(dim(data2))){** *# reshape2 and matrix*

**tempo.cat <- paste0("ERROR IN ", function.name, ": DATA FRAME IN data2 MUST HAVE ROW NUMBER EQUAL TO PRODUCT OF DIMENSIONS OF data1 MATRIX. HERE IT IS RESPECTIVELY:\n", paste(nrow(data1), collapse = " "), "\n", paste(prod(dim(data2)), collapse = " "))**

**text.check <- c(text.check, tempo.cat)**

**arg.check <- c(arg.check, TRUE)**

**}**

**}else if(all(is.data.frame(data2))){**

**tempo <- fun\_check(data = data2, class = "data.frame", length = 3, fun.name = function.name) ; eval(ee)**

**if(tempo$problem == FALSE){**

*# structure of reshape2::melt() data frame*

**tempo <- fun\_check(data = data2[, 1], data.name = "COLUMN 1 OF data2 (reshape2::melt() DATA FRAME)", typeof = "integer", fun.name = function.name) ; eval(ee)**

**tempo <- fun\_check(data = data2[, 2], data.name = "COLUMN 2 OF data2 (reshape2::melt() DATA FRAME)", typeof = "integer", fun.name = function.name) ; eval(ee)**

**tempo <- fun\_check(data = data2[, 3], data.name = "COLUMN 3 OF data2 (reshape2::melt() DATA FRAME)", mode = "numeric", fun.name = function.name) ; eval(ee)**

**}**

**if(tempo$problem == FALSE & ! all(unique(data2[, 3]) %in% c(0,1))){**

**tempo.cat <- paste0("ERROR IN ", function.name, ": THIRD COLUMN OF DATA FRAME IN data2 MUST BE MADE OF 0 AND 1 ONLY (MASK DATA FRAME)")**

**text.check <- c(text.check, tempo.cat)**

**arg.check <- c(arg.check, TRUE)**

**}else if(tempo$problem == FALSE & all(is.data.frame(data1)) & ! identical(dim(data1), dim(data2))){** *# data frame and data frame*

**tempo.cat <- paste0("ERROR IN ", function.name, ": DATA FRAME DIMENSION IN data2 MUST BE IDENTICAL TO DATA FRAME DIMENSION IN data1. HERE IT IS RESPECTIVELY:\n", paste(dim(data2), collapse = " "), "\n", paste(dim(data1), collapse = " "))**

**text.check <- c(text.check, tempo.cat)**

**arg.check <- c(arg.check, TRUE)**

**}else if(tempo$problem == FALSE & all(is.matrix(data1)) & nrow(data2) != prod(dim(data1))){** *# reshape2 and matrix*

**tempo.cat <- paste0("ERROR IN ", function.name, ": DATA FRAME IN data2 MUST HAVE ROW NUMBER EQUAL TO PRODUCT OF DIMENSION OF data1 MATRIX. HERE IT IS RESPECTIVELY:\n", paste(nrow(data2), collapse = " "), "\n", paste(prod(dim(data1)), collapse = " "))**

**text.check <- c(text.check, tempo.cat)**

**arg.check <- c(arg.check, TRUE)**

**}**

**}else{**

**tempo.cat <- paste0("ERROR IN ", function.name, ": THE data2 ARGUMENT MUST BE A NUMERIC MATRIX OR A DATA FRAME OUTPUT OF THE reshape::melt() FUNCTION")**

**text.check <- c(text.check, tempo.cat)**

**arg.check <- c(arg.check, TRUE)**

**}**

**}**

**tempo <- fun\_check(data = color2, class = "character", length = 1, fun.name = function.name) ; eval(ee)**

**if(tempo$problem == FALSE & ! (all(color2 %in% colors() | grepl(pattern = "^#", color2)))){** *# check that all strings of color2 start by #*

**tempo.cat <- paste0("ERROR IN ", function.name, ": color2 ARGUMENT MUST BE A HEXADECIMAL COLOR VECTOR STARTING BY # AND/OR COLOR NAMES GIVEN BY colors()")**

**text.check <- c(text.check, tempo.cat)**

**arg.check <- c(arg.check, TRUE)**

**}**

**tempo <- fun\_check(data = alpha2, class = "vector", mode = "numeric", length = 1, prop = TRUE, fun.name = function.name) ; eval(ee)**

**tempo <- fun\_check(data = invert2, class = "logical", length = 1, fun.name = function.name) ; eval(ee)**

**tempo <- fun\_check(data = text.size, class = "vector", mode = "numeric", length = 1, fun.name = function.name) ; eval(ee)**

**tempo <- fun\_check(data = title, class = "character", length = 1, fun.name = function.name) ; eval(ee)**

**tempo <- fun\_check(data = title.text.size, class = "vector", mode = "numeric", length = 1, fun.name = function.name) ; eval(ee)**

**tempo <- fun\_check(data = show.scale, class = "logical", length = 1, fun.name = function.name) ; eval(ee)**

**tempo <- fun\_check(data = return, class = "logical", length = 1, fun.name = function.name) ; eval(ee)**

**tempo <- fun\_check(data = plot, class = "logical", length = 1, fun.name = function.name) ; eval(ee)**

**if( ! is.null(add)){**

**tempo <- fun\_check(data = add, class = "vector", mode = "character", length = 1, fun.name = function.name) ; eval(ee)**

**if(tempo$problem == FALSE & ! grepl(pattern = "^\\+", add)){** *# check that the add string start by +*

**tempo.cat <- paste0("ERROR IN ", function.name, ": add ARGUMENT MUST START WITH \"+\": ", paste(unique(add), collapse = " "))**

**text.check <- c(text.check, tempo.cat)**

**arg.check <- c(arg.check, TRUE)**

**}else if(tempo$problem == FALSE & ! grepl(pattern = "ggplot2::", add)){** *#*

**tempo.cat <- paste0("ERROR IN ", function.name, ": add ARGUMENT MUST CONTAIN \"ggplot2::\" IN FRONT OF EACH GGPLOT2 FUNCTION: ", paste(unique(add), collapse = " "))**

**text.check <- c(text.check, tempo.cat)**

**arg.check <- c(arg.check, TRUE)**

**}else if(tempo$problem == FALSE & ! grepl(pattern = ")$", add)){** *# check that the add string finished by )*

**tempo.cat <- paste0("ERROR IN ", function.name, ": add ARGUMENT MUST FINISH BY \")\": ", paste(unique(add), collapse = " "))**

**text.check <- c(text.check, tempo.cat)**

**arg.check <- c(arg.check, TRUE)**

**}**

**}**

**tempo <- fun\_check(data = warn.print, class = "logical", length = 1, fun.name = function.name) ; eval(ee)**

**if( ! is.null(lib.path)){**

**tempo <- fun\_check(data = lib.path, class = "vector", mode = "character", fun.name = function.name) ; eval(ee)**

**if(tempo$problem == FALSE){**

**if( ! all(dir.exists(lib.path))){** *# separation to avoid the problem of tempo$problem == FALSE and lib.path == NA*

**tempo.cat <- paste0("ERROR IN ", function.name, ": DIRECTORY PATH INDICATED IN THE lib.path ARGUMENT DOES NOT EXISTS:\n", paste(lib.path, collapse = "\n"))**

**text.check <- c(text.check, tempo.cat)**

**arg.check <- c(arg.check, TRUE)**

**}**

**}**

**}**

**if(any(arg.check) == TRUE){**

**stop(paste0("\n\n================\n\n", paste(text.check[arg.check], collapse = "\n"), "\n\n================\n\n"), call. = FALSE)** *#*

**}**

*# source("C:/Users/Gael/Documents/Git\_versions\_to\_use/debugging\_tools\_for\_r\_dev-v1.2/r\_debugging\_tools-v1.2.R") ; eval(parse(text = str\_basic\_arg\_check\_dev)) ; eval(parse(text = str\_arg\_check\_with\_fun\_check\_dev)) # activate this line and use the function (with no arguments left as NULL) to check arguments status and if they have been checked using fun\_check()*

*# end argument checking*

*# package checking*

**fun\_pack(req.package = c("reshape2", "ggplot2"), lib.path = lib.path)**

*# end package checking*

*# main code*

**ini.warning.length <- options()$warning.length**

**options(warning.length = 8170)**

**on.exit(exp = options(warning.length = ini.warning.length))**

**warn <- NULL**

**warn.count <- 0**

**if(all(is.matrix(data1))){**

**data1 <- reshape2::melt(data1)** *# transform a matrix into a data frame with 2 coordinates columns and the third intensity column*

**}**

**if(rotate == TRUE){**

**data1[, 1] <- rev(data1[, 1])**

**}**

**if(is.null(limit1)){**

**if(any(data1[, 3] %in% c(Inf, -Inf))){**

**warn.count <- warn.count + 1**

**tempo.warn <- paste0("(", warn.count,") THE data1 ARGUMENT CONTAINS -Inf OR Inf VALUES IN THE THIRD COLUMN, THAT WILL NOT BE CONSIDERED IN THE PLOT RANGE")**

**warn <- paste0(ifelse(is.null(warn), tempo.warn, paste0(warn, "\n\n", tempo.warn)))**

**}**

**limit1 <- range(data1[, 3], na.rm = TRUE, finite = TRUE)** *# finite = TRUE removes all the -Inf and Inf except if only this. In that case, whatever the -Inf and/or Inf present, output -Inf;Inf range. Idem with NA only*

**warn.count <- warn.count + 1**

**tempo.warn <- paste0("(", warn.count,") THE limit1 ARGUMENT IS NULL -> RANGE OF data1 ARGUMENT HAS BEEN TAKEN: ", paste(fun\_round(limit1), collapse = " "))**

**warn <- paste0(ifelse(is.null(warn), tempo.warn, paste0(warn, "\n\n", tempo.warn)))**

**if(suppressWarnings(any(limit1 %in% c(Inf, -Inf)))){**

**tempo.cat <- paste0("ERROR IN ", function.name, " COMPUTED LIMIT CONTAINS Inf VALUES, BECAUSE VALUES FROM data1 ARGUMENTS ARE NA OR Inf ONLY")**

**stop(paste0("\n\n================\n\n", tempo.cat, "\n\n================\n\n", ifelse(is.null(warn), "", paste0("IN ADDITION\nWARNING", ifelse(warn.count > 1, "S", ""), ":\n\n", warn))), call. = FALSE)** *# == in stop() to be able to add several messages between ==*

**}**

**}**

**if(is.null(midpoint1)){**

**midpoint1 <- mean(limit1, na.rm = TRUE)**

**warn.count <- warn.count + 1**

**tempo.warn <- paste0("(", warn.count,") THE midpoint1 ARGUMENT IS NULL -> MEAN OF limit1 ARGUMENT HAS BEEN TAKEN: ", paste(fun\_round(midpoint1), collapse = " "))**

**warn <- paste0(ifelse(is.null(warn), tempo.warn, paste0(warn, "\n\n", tempo.warn)))**

**}else if(fun\_round(midpoint1, 9) != fun\_round(mean(limit1), 9)){**

**warn.count <- warn.count + 1**

**tempo.warn <- paste0("(", warn.count,") THE midpoint1 ARGUMENT (", fun\_round(mean(midpoint1), 9), ") DOES NOT CORRESPOND TO THE MEAN OF THE limit1 ARGUMENT (", fun\_round(mean(limit1), 9), "). COLOR SCALE IS NOT LINEAR")**

**warn <- paste0(ifelse(is.null(warn), tempo.warn, paste0(warn, "\n\n", tempo.warn)))**

**}**

**if( ! is.null(data2)){**

**if(all(is.matrix(data2))){**

**data2 <- reshape2::melt(data2)** *# transform a matrix into a data frame with 2 coordinates columns and the third intensity column*

**}**

**if(rotate == TRUE){**

**data2[, 1] <- rev(data2[, 1])**

**}**

**data2[, 3] <- factor(data2[, 3])** *# to converte continuous scale into discrete scale*

**}**

**tempo.gg.name <- "gg.indiv.plot."**

**tempo.gg.count <- 0** *# to facilitate debugging*

**assign(paste0(tempo.gg.name, tempo.gg.count <- tempo.gg.count + 1), ggplot2::ggplot())**

**assign(paste0(tempo.gg.name, tempo.gg.count <- tempo.gg.count + 1), ggplot2::geom\_raster(data = data1, mapping = ggplot2::aes\_string(x = names(data1)[ifelse(rotate == FALSE, 2, 1)], y = names(data1)[ifelse(rotate == FALSE, 1, 2)], fill = names(data1)[3]), show.legend = show.scale))** *# show.legend option do not remove the legend, only the aesthetic of the legend (dot, line, etc.)*

**assign(paste0(tempo.gg.name, tempo.gg.count <- tempo.gg.count + 1), ggplot2::scale\_fill\_gradient2(low = low.color1, high = high.color1, mid = mid.color1, midpoint = midpoint1, limit = limit1, breaks = c(limit1[1], midpoint1, limit1[2]), labels = fun\_round(c(limit1[1], midpoint1, limit1[2])), name = legend.name1))**

**if( ! is.null(data2)){**

**assign(paste0(tempo.gg.name, tempo.gg.count <- tempo.gg.count + 1), ggplot2::geom\_raster(data = data2, mapping = ggplot2::aes\_string(x = names(data2)[ifelse(rotate == FALSE, 2, 1)], y = names(data2)[ifelse(rotate == FALSE, 1, 2)], alpha = names(data2)[3]), fill = color2, show.legend = FALSE))**

**assign(paste0(tempo.gg.name, tempo.gg.count <- tempo.gg.count + 1), ggplot2::scale\_discrete\_manual(aesthetics = "alpha", values = if(invert2 == FALSE){c(0, alpha2)}else{c(alpha2, 0)}, guide = FALSE))**

*# assign(paste0(tempo.gg.name, tempo.gg.count <- tempo.gg.count + 1), ggplot2::geom\_raster(data = data2, mapping = ggplot2::aes\_string(x = names(data2)[ifelse(rotate == FALSE, 2, 1)], y = names(data2)[ifelse(rotate == FALSE, 1, 2)], group = names(data2)[3]), fill = data2[, 3], alpha = alpha2, show.legend = FALSE)) # BEWARE: this does not work if NA present, because geom\_raster() has a tendency to complete empty spaces, and thus, behave differently than geom\_tile(). See https://github.com/tidyverse/ggplot2/issues/3025*

**}**

**assign(paste0(tempo.gg.name, tempo.gg.count <- tempo.gg.count + 1), ggplot2::coord\_fixed())** *# x = y*

**assign(paste0(tempo.gg.name, tempo.gg.count <- tempo.gg.count + 1), ggplot2::scale\_y\_reverse())**

**assign(paste0(tempo.gg.name, tempo.gg.count <- tempo.gg.count + 1), ggplot2::ggtitle(title))**

**add.check <- TRUE**

**if( ! is.null(add)){** *# if add is NULL, then = 0*

**if(grepl(pattern = "ggplot2::theme", add) == TRUE){**

**warn.count <- warn.count + 1**

**tempo.warn <- paste0("(", warn.count,") \"ggplot2::theme\" STRING DETECTED IN THE add ARGUMENT -> INTERNAL GGPLOT2 THEME FUNCTIONS theme() AND theme\_classic() HAVE BEEN INACTIVATED, TO BE USED BY THE USER")**

**warn <- paste0(ifelse(is.null(warn), tempo.warn, paste0(warn, "\n\n", tempo.warn)))**

**add.check <- FALSE**

**}**

**}**

**if(add.check == TRUE){**

**assign(paste0(tempo.gg.name, tempo.gg.count <- tempo.gg.count + 1), ggplot2::theme\_classic(base\_size = text.size))**

**assign(paste0(tempo.gg.name, tempo.gg.count <- tempo.gg.count + 1), ggplot2::theme(**

**text = ggplot2::element\_text(size = text.size),**

**plot.title = ggplot2::element\_text(size = title.text.size),** *# stronger than text*

**line = ggplot2::element\_blank(),**

**axis.title = ggplot2::element\_blank(),**

**axis.text = ggplot2::element\_blank(),**

**axis.ticks = ggplot2::element\_blank(),**

**panel.background = ggplot2::element\_blank()**

**))**

**}**

**if(plot == TRUE){**

**# suppressWarnings(**

**print(eval(parse(text = paste(paste(paste0(tempo.gg.name, 1:tempo.gg.count), collapse = " + "), if(is.null(add)){NULL}else{add}))))**

**# )**

**}else{**

**warn.count <- warn.count + 1**

**tempo.warn <- paste0("(", warn.count,") PLOT NOT SHOWN AS REQUESTED")**

**warn <- paste0(ifelse(is.null(warn), tempo.warn, paste0(warn, "\n\n", tempo.warn)))**

**}**

**if(warn.print == TRUE & ! is.null(warn)){**

**warning(paste0("FROM ", function.name, ":\n\n", warn), call. = FALSE)**

**}**

**if(return == TRUE){**

**output <- ggplot2::ggplot\_build(eval(parse(text = paste(paste0(tempo.gg.name, 1:tempo.gg.count), collapse = " + "))))**

**output <- output$data**

**names(output)[1] <- "heatmap"**

**if( ! is.null(data2)){**

**names(output)[2] <- "mask"**

**}**

**return(list(data = output, axes = output$layout$panel\_params[[1]], scale = c(limit1[1], midpoint1, limit1[2]), warn = warn))**

**}**

**}**

######## fun\_gg\_empty\_graph() #### text to display for empty graphs



**# Check OK: clear to go Apollo**

**fun\_gg\_empty\_graph <- function(text = NULL, text.size = 12, title = NULL, title.size = 8, lib.path = 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*

*# title.size: numeric value of the title size (in points)*

*# lib.path: character vector specifying the absolute pathways of the directories containing the required packages if not in the default directories. Ignored if NULL*

*# REQUIRED PACKAGES*

*# ggplot2*

*# REQUIRED FUNCTIONS FROM CUTE\_LITTLE\_R\_FUNCTION*

*# fun\_check()*

*# fun\_pack()*

*# RETURN*

*# an empty plot*

*# EXAMPLES*

*### simple example*

*# fun\_gg\_empty\_graph(text = "NO GRAPH")*

*### white page*

*# fun\_gg\_empty\_graph()*

*### all the arguments*

*# fun\_gg\_empty\_graph(text = "NO GRAPH", text.size = 8, title = "GRAPH1", title.size = 10, lib.path = NULL)*

*# DEBUGGING*

*# text = "NO GRAPH" ; text.size = 12 ; title = "GRAPH1" ; title.size = 8 ; lib.path = NULL*

*# function name*

**function.name <- paste0(as.list(match.call(expand.dots=FALSE))[[1]], "()")**

*# end function name*

*# required function checking*

**if(length(utils::find("fun\_check", mode = "function")) == 0){**

**tempo.cat <- paste0("ERROR IN ", function.name, ": REQUIRED fun\_check() FUNCTION IS MISSING IN THE R ENVIRONMENT")**

**stop(paste0("\n\n================\n\n", tempo.cat, "\n\n================\n\n"), call. = FALSE)** *# == in stop() to be able to add several messages between ==*

**}**

**if(length(utils::find("fun\_pack", mode = "function")) == 0){**

**tempo.cat <- paste0("ERROR IN ", function.name, ": REQUIRED fun\_pack() FUNCTION IS MISSING IN THE R ENVIRONMENT")**

**stop(paste0("\n\n================\n\n", tempo.cat, "\n\n================\n\n"), call. = FALSE)** *# == in stop() to be able to add several messages between ==*

**}**

*# end required function checking*

*# argument checking*

**arg.check <- NULL** *#*

**text.check <- NULL** *#*

**checked.arg.names <- NULL** *# for function debbuging: used by r\_debugging\_tools*

**ee <- expression(arg.check <- c(arg.check, tempo$problem) , text.check <- c(text.check, tempo$text) , checked.arg.names <- c(checked.arg.names, tempo$fun.name))**

**if( ! is.null(text)){**

**tempo <- fun\_check(data = text, class = "vector", mode = "character", length = 1, fun.name = function.name) ; eval(ee)**

**}**

**tempo <- fun\_check(data = text.size, class = "vector", mode = "numeric", length = 1, fun.name = function.name) ; eval(ee)**

**if( ! is.null(title)){**

**tempo <- fun\_check(data = title, class = "vector", mode = "character", length = 1, fun.name = function.name) ; eval(ee)**

**}**

**tempo <- fun\_check(data = title.size, class = "vector", mode = "numeric", length = 1, fun.name = function.name) ; eval(ee)**

**if(any(arg.check) == TRUE){**

**stop(paste0("\n\n================\n\n", paste(text.check[arg.check], collapse = "\n"), "\n\n================\n\n"), call. = FALSE)** *#*

**}**

*# source("C:/Users/Gael/Documents/Git\_versions\_to\_use/debugging\_tools\_for\_r\_dev-v1.2/r\_debugging\_tools-v1.2.R") ; eval(parse(text = str\_basic\_arg\_check\_dev)) ; eval(parse(text = str\_arg\_check\_with\_fun\_check\_dev)) # activate this line and use the function (with no arguments left as NULL) to check arguments status and if they have been checked using fun\_check()*

*# end argument checking*

*# package checking*

**fun\_pack(req.package = c("ggplot2"), lib.path = lib.path)**

*# 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())**

**if( ! is.null(text)){**

**assign(paste0(tempo.gg.name, tempo.gg.count <- tempo.gg.count + 1), ggplot2::geom\_text(data = data.frame(x = 1, y = 1), ggplot2::aes(x = x, y = y, label = text), size = text.size))**

**}**

**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())**

**assign(paste0(tempo.gg.name, tempo.gg.count <- tempo.gg.count + 1), m.gg <- ggplot2::theme(**

**plot.title = ggplot2::element\_text(size = title.size)** *# stronger than text*

**))**

**suppressWarnings(print(eval(parse(text = paste(paste0(tempo.gg.name, 1:tempo.gg.count), collapse = " + ")))))**

**}**

################ Graphic extraction

######## fun\_trim() #### display values from a quantitative variable and trim according to defined cut-offs

**# Check OK: clear to go Apollo**

**fun\_trim <- 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, corner.text.size = 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, histogram 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\_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*

*# corner.text.size: positive numeric. Increase or decrease the size of the text. Value 1 does not change it, 0.5 decreases by half, 2 increases by 2*

*# 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\_trim(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, corner.text.size = 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 ; corner.text.size = 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(utils::find("fun\_check", mode = "function")) == 0){**

**tempo.cat <- paste0("ERROR IN ", function.name, ": REQUIRED fun\_check() FUNCTION IS MISSING IN THE R ENVIRONMENT")**

**stop(paste0("\n\n================\n\n", tempo.cat, "\n\n================\n\n"), call. = FALSE)** *# == in stop() to be able to add several messages between ==*

**}**

*# end required function checking*

*# argument checking*

*# argument checking without fun\_check()*

**if( ! (all(class(data) == "numeric") | all(class(data) == "integer") | (all(class(data) %in% c("matrix", "array")) & mode(data) == "numeric"))){**

**tempo.cat <- paste0("ERROR IN ", function.name, ": data ARGUMENT MUST BE A NUMERIC VECTOR OR NUMERIC MATRIX")**

**stop(paste0("\n\n================\n\n", tempo.cat, "\n\n================\n\n"), call. = FALSE)** *# == in stop() to be able to add several messages between ==*

**}**

*# end argument checking without fun\_check()*

*# argument checking with fun\_check()*

**arg.check <- NULL** *#*

**text.check <- NULL** *#*

**checked.arg.names <- NULL** *# for function debbuging: used by r\_debugging\_tools*

**ee <- expression(arg.check <- c(arg.check, tempo$problem) , text.check <- c(text.check, tempo$text) , checked.arg.names <- c(checked.arg.names, tempo$fun.name))**

**if( ! is.null(displayed.nb)){**

**tempo <- fun\_check(data = displayed.nb, class = "vector", mode = "numeric", length = 1, fun.name = function.name) ; eval(ee)**

**if(displayed.nb < 2){**

**tempo.cat <- paste0("ERROR IN ", function.name, ": displayed.nb ARGUMENT MUST BE A SINGLE INTEGER VALUE GREATER THAN 1 AND NOT: ", paste(displayed.nb, collapse = " "))**

**text.check <- c(text.check, tempo.cat)**

**arg.check <- c(arg.check, TRUE)**

**}**

**}**

**tempo <- fun\_check(data = single.value.display, class = "logical", length = 1, fun.name = function.name) ; eval(ee)**

**tempo <- fun\_check(data = trim.method, options = c("", "mean.sd", "quantile"), length = 1, fun.name = function.name) ; eval(ee)**

**tempo <- fun\_check(data = trim.cutoffs, class = "vector", mode = "numeric", length = 2, prop = TRUE, fun.name = function.name) ; eval(ee)**

**tempo <- fun\_check(data = interval.scale.disp, class = "logical", length = 1, fun.name = function.name) ; eval(ee)**

**tempo <- fun\_check(data = down.space, class = "vector", mode = "numeric", length = 1, neg.values = FALSE, fun.name = function.name) ; eval(ee)**

**tempo <- fun\_check(data = left.space, class = "vector", mode = "numeric", length = 1, neg.values = FALSE, fun.name = function.name) ; eval(ee)**

**tempo <- fun\_check(data = up.space, class = "vector", mode = "numeric", length = 1, neg.values = FALSE, fun.name = function.name) ; eval(ee)**

**tempo <- fun\_check(data = right.space, class = "vector", mode = "numeric", length = 1, neg.values = FALSE, fun.name = function.name) ; eval(ee)**

**tempo <- fun\_check(data = orient, class = "vector", mode = "numeric", length = 1, neg.values = FALSE, fun.name = function.name) ; eval(ee)**

**tempo <- fun\_check(data = dist.legend, class = "vector", mode = "numeric", length = 1, neg.values = FALSE, fun.name = function.name) ; eval(ee)**

**tempo <- fun\_check(data = box.type, options = c("o", "l", "7", "c", "u", "]", "n"), length = 1, fun.name = function.name) ; eval(ee)**

**tempo <- fun\_check(data = amplif.label, class = "vector", mode = "numeric", length = 1, neg.values = FALSE, fun.name = function.name) ; eval(ee)**

**tempo <- fun\_check(data = amplif.axis, class = "vector", mode = "numeric", length = 1, neg.values = FALSE, fun.name = function.name) ; eval(ee)**

**tempo <- fun\_check(data = std.x.range, class = "logical", length = 1, fun.name = function.name) ; eval(ee)**

**tempo <- fun\_check(data = std.y.range, class = "logical", length = 1, fun.name = function.name) ; eval(ee)**

**tempo <- fun\_check(data = cex.pt, class = "vector", mode = "numeric", length = 1, neg.values = FALSE, fun.name = function.name) ; eval(ee)**

**tempo <- fun\_check(data = col.box, class = "character", length = 1, fun.name = function.name) ; eval(ee)**

**tempo <- fun\_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\_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\_check(data = tick.length, class = "vector", mode = "numeric", length = 1, prop = TRUE, fun.name = function.name) ; eval(ee)**

**tempo <- fun\_check(data = sec.tick.length, class = "vector", mode = "numeric", length = 1, prop = TRUE, fun.name = function.name) ; eval(ee)**

**tempo <- fun\_check(data = corner.text, class = "character", length = 1, fun.name = function.name) ; eval(ee)**

**tempo <- fun\_check(data = amplif.legend, class = "vector", mode = "numeric", length = 1, neg.values = FALSE, fun.name = function.name) ; eval(ee)**

**tempo <- fun\_check(data = corner.text.size, class = "vector", mode = "numeric", length = 1, neg.values = FALSE, fun.name = function.name) ; eval(ee)**

**tempo <- fun\_check(data = trim.return, class = "logical", length = 1, fun.name = function.name) ; eval(ee)**

**if(any(arg.check) == TRUE){**

**stop(paste0("\n\n================\n\n", paste(text.check[arg.check], collapse = "\n"), "\n\n================\n\n"), call. = FALSE)** *#*

**}**

*# end argument checking with fun\_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\_check\_dev)) # activate this line and use the function (with no arguments left as NULL) to check arguments status and if they have been checked using fun\_check()*

**if(all(is.na(data) | ! is.finite(data))){**

**tempo.cat <- paste0("ERROR IN fun\_trim FUNCTION\ndata ARGUMENT CONTAINS ONLY NA OR Inf")**

**stop(paste0("\n\n================\n\n", tempo.cat, "\n\n================\n\n"), call. = FALSE)** *# == in stop() to be able to add several messages between ==*

**}**

*# end argument checking*

*# main code*

**if(class(data)%in% c("matrix", "array")){**

**data <- as.vector(data)**

**}**

**na.nb <- NULL**

**if(any(is.na(data))){**

**na.nb <- sum(c(is.na(data)))**

**data <- data[ ! is.na(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, na.rm = TRUE)**

**abline(v = quantile(data.f, probs = trim.cutoffs.f, type = 7, na.rm = TRUE), col = color.cut.f)**

**segments(quantile(data.f, probs = trim.cutoffs.f[1], type = 7, na.rm = TRUE), par()$usr[4] \* 0.75, quantile(data.f, probs = trim.cutoffs.f[2], type = 7, na.rm = TRUE), par()$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, na.rm = TRUE), 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("WARNING: ONLY ", displayed.nb, " VALUES ARE DISPLAYED AMONG THE ", length(as.vector(data)), " VALUES OF THE DATASET ANALYZED")**

**}else{**

**corner.text <- paste0(corner.text, "\nWARNING: 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("WARNING: 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, "\nWARNING: 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")**

**}**

**}**

**}**

**if( ! is.null(na.nb)){**

**if(corner.text == ""){**

**corner.text <- paste0("WARNING: NUMBER OF NA REMOVED IS ", na.nb)**

**}else{**

**corner.text <- paste0("WARNING: NUMBER OF NA REMOVED IS ", na.nb)**

**}**

**}**

**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 = "", xlim = graph.xlim, xlab = "Value", ylab="Density", col = grey(0.25))** *# removed: breaks = seq(min(as.vector(data), na.rm = TRUE), max(as.vector(data), na.rm = TRUE), length.out = length(as.vector(data)) / 10)*

**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 = corner.text.size)** *# 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, data2.pb.dot = "unknown", xy.cross.kind = "&", plot = FALSE, graph.in.file = FALSE, raster = TRUE, warn.print = FALSE, lib.path = 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*

*# WARNINGS*

*# if dots from data2 look significant on the graph (outside the frame) but are not (not black on the last figure), this is probably because the frame is flat on the zero coordinate (no volume inside the frame at this position). Thus, no way to conclude that data2 dots here are significant. These dots are refered to as "unknown". The pb.dot argument deals with such dots*

*# dots that are sometimes inside and outside the frame, depending on the sliding window, are treated differently: they are removed. Such dots are neither classified as "signif", "non signif" or "unknown", but as "inconsistent"*

*# unknown dots are treated as finally significant, not significant, or unknown (data2.pb.dot argument) for each x-axis and y-axis separately. Then, the union or intersection of significant dots is performed (argument xy.cross.kind). See the example section*

*# ARGUMENTS*

*# data1: a data frame 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. Write NULL if not required. 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 data frame 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)*

*# data2.pb.dot: unknown dots are explain in the warning section above. If "signif", then the unknown dots are finally considered as significant (outside the frame). If "not.signif", then the unknown dots are finally considered as non significant (inside the frame). If "unknown", no conclusion are drawn from these dots. See the examples below*

*# 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. See the examples below*

*# plot: logical. Print graphs that check the frame?*

*# graph.in.file: logical. Graphs sent into a graphic device already opened? If FALSE, GUI are opened for each graph. If TRUE, no GUI are opended. The graphs are displayed on the current active graphic device. Ignored if plot is FALSE*

*# raster: logical. Dots in raster mode? If FALSE, dots from each geom\_point from geom argument are in vectorial mode (bigger pdf and long to display if millions of dots). If TRUE, dots from each geom\_point from geom argument are in matricial mode (smaller pdf and easy display if millions of dots, but long to generate the layer). If TRUE, the region plot will be square to avoid a bug in fun\_gg\_point\_rast(). If TRUE, solve the transparency problem with some GUI. Not considered if plot is FALSE*

*# warn.print: logical. Print warnings at the end of the execution? No print if no warning messages*

*# lib.path: character vector specifying the absolute pathways of the directories containing the required packages if not in the default directories. Ignored if NULL. Ignored if plot is FALSE*

*# REQUIRED PACKAGES*

*# ggplot2 if plot is TRUE*

*# REQUIRED FUNCTIONS FROM CUTE\_LITTLE\_R\_FUNCTION*

*# fun\_check()*

*# if plot is TRUE:*

*# fun\_pack()*

*# fun\_open()*

*# fun\_gg\_palette()*

*# fun\_gg\_scatter()*

*# fun\_gg\_empty\_graph()*

*# fun\_close()*

*# RETURN*

*# several graphs if plot is TRUE*

*# a list containing:*

*# $data1.removed.row.nb: which rows have been removed due to NA; NaN, -Inf or Inf detection in x1 or y1 columns (NULL if no row removed)*

*# $data1.removed.rows: removed rows (NULL if no row removed)*

*# $data2.removed.row.nb: which rows have been removed due to NA; NaN, -Inf or Inf detection in x2 or y2 columns (NULL if no row removed)*

*# $data2.removed.rows: removed rows (NULL if no row removed)*

*# $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). A good segmentation should not have any data1.signif.dot*

*# $data1.non.signif.dot: the non significant dots of data1 (i.e., dots inside the frame)*

*# $data1.inconsistent.dot: see the warning section above*

*# $data2.signif.dot: the significant dots of data2 if non NULL (i.e., dots outside the frame)*

*# $data2.non.signif.dot: the non significant dots of data2 (i.e., dots inside the frame)*

*# $data2.unknown.dot: the problematic dots of data2 (i.e., data2 dots outside of the range of data1, or data2 dots in a sliding window without data1 dots). Is systematically NULL except if argument data2.pb.dot = "unknown" and some data2 dots are in such situation. Modifying the segmentation x.range.split, x.step.factor, y.range.split, y.step.factor arguments can solve this problem*

*# $data2.inconsistent.dot: see the warning section above*

*# $axes: the x-axis and y-axis info*

*# $warn: the warning messages. Use cat() for proper display. NULL if no warning*

*# EXAMPLES*

*# example explaining the unknown and inconsistent dots, and the cross*

*# set.seed(1) ; data1 = data.frame(x = rnorm(500), y = rnorm(500)) ; data1[5:7, 2] <- NA ; data2 = data.frame(x = rnorm(500, 0, 2), y = rnorm(500, 0, 2)) ; data2[11:13, 1] <- Inf ; 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], data2.pb.dot = "not.signif", xy.cross.kind = "|", plot = TRUE, graph.in.file = FALSE, raster = FALSE, lib.path = 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], data2.pb.dot = "unknown", xy.cross.kind = "|", plot = TRUE, graph.in.file = FALSE, raster = FALSE, lib.path = 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], data2.pb.dot = "unknown", xy.cross.kind = "&", plot = TRUE, graph.in.file = FALSE, raster = FALSE, lib.path = NULL)*

*# DEBUGGING*

*# set.seed(1) ; data1 = data.frame(x = rnorm(50), y = rnorm(50)) ; data1[5:7, 2] <- NA ; 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] ; data2.pb.dot = "unknown" ; xy.cross.kind = "|" ; plot = TRUE ; graph.in.file = FALSE ; raster = FALSE ; warn.print = TRUE ; lib.path = 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 = 23 ; y.step.factor = 10 ; error = 0 ; x2 = names(data2)[1] ; y2 = names(data2)[2] ; data2.pb.dot = "not.signif" ; xy.cross.kind = "|" ; plot = TRUE ; graph.in.file = FALSE ; raster = FALSE ; warn.print = TRUE ; lib.path = 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] ; data2.pb.dot = "unknown" ; xy.cross.kind = "&" ; plot = TRUE ; graph.in.file = FALSE ; raster = FALSE ; warn.print = TRUE ; lib.path = NULL*

*# function name*

**function.name <- paste0(as.list(match.call(expand.dots=FALSE))[[1]], "()")**

*# end function name*

*# required function checking*

**if(length(utils::find("fun\_check", mode = "function")) == 0){**

**tempo.cat <- paste0("ERROR IN ", function.name, ": REQUIRED fun\_check() FUNCTION IS MISSING IN THE R ENVIRONMENT")**

**stop(paste0("\n\n================\n\n", tempo.cat, "\n\n================\n\n"), call. = FALSE)** *# == in stop() to be able to add several messages between ==*

**}**

*# end required function checking*

*# argument checking*

**ini.warning.length <- options()$warning.length**

**options(warning.length = 8170)**

**on.exit(exp = options(warning.length = ini.warning.length))**

**warn <- NULL**

**warn.count <- 0**

**arg.check <- NULL** *#*

**text.check <- NULL** *#*

**checked.arg.names <- NULL** *# for function debbuging: used by r\_debugging\_tools*

**ee <- expression(arg.check <- c(arg.check, tempo$problem) , text.check <- c(text.check, tempo$text) , checked.arg.names <- c(checked.arg.names, tempo$fun.name))**

**tempo <- fun\_check(data = data1, class = "data.frame", na.contain = TRUE, fun.name = function.name) ; eval(ee)**

**if(tempo$problem == FALSE & length(data1) < 2){**

**tempo.cat <- paste0("ERROR IN ", function.name, ": data1 ARGUMENT MUST BE A DATA FRAME OF AT LEAST 2 COLUMNS")**

**text.check <- c(text.check, tempo.cat)**

**arg.check <- c(arg.check, TRUE)**

**}**

**tempo <- fun\_check(data = x1, class = "vector", mode = "character", length = 1, fun.name = function.name) ; eval(ee)**

**if(tempo$problem == FALSE & ! (x1 %in% names(data1))){**

**tempo.cat <- paste0("ERROR IN ", function.name, ": x1 ARGUMENT MUST BE A COLUMN NAME OF data1")**

**text.check <- c(text.check, tempo.cat)**

**arg.check <- c(arg.check, TRUE)**

**}else if(tempo$problem == FALSE & x1 %in% names(data1)){**

**tempo <- fun\_check(data = data1[, x1], data.name = "x1 COLUMN OF data1", class = "vector", mode = "numeric", na.contain = TRUE, fun.name = function.name) ; eval(ee)**

**}**

**tempo <- fun\_check(data = y1, class = "vector", mode = "character", length = 1, fun.name = function.name) ; eval(ee)**

**if(tempo$problem == FALSE & ! (y1 %in% names(data1))){**

**tempo.cat <- paste0("ERROR IN ", function.name, ": y1 ARGUMENT MUST BE A COLUMN NAME OF data1")**

**text.check <- c(text.check, tempo.cat)**

**arg.check <- c(arg.check, TRUE)**

**}else if(tempo$problem == FALSE & y1 %in% names(data1)){**

**tempo <- fun\_check(data = data1[, y1], data.name = "y1 COLUMN OF data1", class = "vector", mode = "numeric", na.contain = TRUE, fun.name = function.name) ; eval(ee)**

**}**

**if(is.null(x.range.split) & is.null(y.range.split)){**

**tempo.cat <- paste0("ERROR IN ", function.name, ": AT LEAST ONE OF THE x.range.split AND y.range.split ARGUMENTS MUST BE NON NULL")**

**text.check <- c(text.check, tempo.cat)**

**arg.check <- c(arg.check, TRUE)**

**}**

**if( ! is.null(x.range.split)){**

**tempo <- fun\_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){**

**tempo.cat <- paste0("ERROR IN ", function.name, ": x.range.split ARGUMENT CANNOT BE LOWER THAN 1")**

**text.check <- c(text.check, tempo.cat)**

**arg.check <- c(arg.check, TRUE)**

**}**

**}**

**if( ! is.null(y.range.split)){**

**tempo <- fun\_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){**

**tempo.cat <- paste0("ERROR IN ", function.name, ": y.range.split ARGUMENT CANNOT BE LOWER THAN 1")**

**text.check <- c(text.check, tempo.cat)**

**arg.check <- c(arg.check, TRUE)**

**}**

**}**

**tempo <- fun\_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){**

**tempo.cat <- paste0("ERROR IN ", function.name, ": x.step.factor ARGUMENT CANNOT BE LOWER THAN 1")**

**text.check <- c(text.check, tempo.cat)**

**arg.check <- c(arg.check, TRUE)**

**}**

**tempo <- fun\_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){**

**tempo.cat <- paste0("ERROR IN ", function.name, ": y.step.factor ARGUMENT CANNOT BE LOWER THAN 1")**

**text.check <- c(text.check, tempo.cat)**

**arg.check <- c(arg.check, TRUE)**

**}**

**tempo <- fun\_check(data = error, prop = TRUE, length = 1, fun.name = function.name) ; eval(ee)**

**if( ! is.null(data2)){**

**if(is.null(x2) | is.null(y2)){**

**tempo.cat <- paste0("ERROR IN ", function.name, ": x2 AND y2 ARGUMENTS CANNOT BE NULL IF data2 ARGUMENT IS NON NULL")**

**text.check <- c(text.check, tempo.cat)**

**arg.check <- c(arg.check, TRUE)**

**}**

**tempo <- fun\_check(data = data2, class = "data.frame", na.contain = TRUE, fun.name = function.name) ; eval(ee)**

**if(tempo$problem == FALSE & length(data2) < 2){**

**tempo.cat <- paste0("ERROR IN ", function.name, ": data2 ARGUMENT MUST BE A DATA FRAME OF AT LEAST 2 COLUMNS")**

**text.check <- c(text.check, tempo.cat)**

**arg.check <- c(arg.check, TRUE)**

**}**

**if( ! is.null(x2)){**

**tempo <- fun\_check(data = x2, class = "vector", mode = "character", length = 1, fun.name = function.name) ; eval(ee)**

**if(tempo$problem == FALSE & ! (x2 %in% names(data2))){**

**tempo.cat <- paste0("ERROR IN ", function.name, ": x2 ARGUMENT MUST BE A COLUMN NAME OF data2")**

**text.check <- c(text.check, tempo.cat)**

**arg.check <- c(arg.check, TRUE)**

**}else if(tempo$problem == FALSE & x2 %in% names(data2)){**

**tempo <- fun\_check(data = data2[, x2], data.name = "x2 COLUMN OF data2", class = "vector", mode = "numeric", na.contain = TRUE, fun.name = function.name) ; eval(ee)**

**}**

**}**

**if( ! is.null(y2)){**

**tempo <- fun\_check(data = y2, class = "vector", mode = "character", length = 1, fun.name = function.name) ; eval(ee)**

**if(tempo$problem == FALSE & ! (y2 %in% names(data2))){**

**tempo.cat <- paste0("ERROR IN ", function.name, ": y2 ARGUMENT MUST BE A COLUMN NAME OF data2")**

**text.check <- c(text.check, tempo.cat)**

**arg.check <- c(arg.check, TRUE)**

**}else if(tempo$problem == FALSE & y2 %in% names(data2)){**

**tempo <- fun\_check(data = data2[, y2], data.name = "y2 COLUMN OF data2", class = "vector", mode = "numeric", na.contain = TRUE, fun.name = function.name) ; eval(ee)**

**}**

**}**

**}**

**if( ! is.null(data2)){**

**tempo <- fun\_check(data = data2.pb.dot, options = c("signif", "not.signif", "unknown"), length = 1, fun.name = function.name) ; eval(ee)**

**}**

**if( ! (is.null(x.range.split)) & ! (is.null(y.range.split))){**

**tempo <- fun\_check(data = xy.cross.kind, options = c("&", "|"), length = 1, fun.name = function.name) ; eval(ee)**

**}**

**tempo <- fun\_check(data = plot, class = "vector", mode = "logical", length = 1, fun.name = function.name) ; eval(ee)**

**tempo <- fun\_check(data = warn.print, class = "logical", length = 1, fun.name = function.name) ; eval(ee)**

**if(tempo$problem == FALSE & plot == TRUE){**

**tempo <- fun\_check(data = raster, class = "vector", mode = "logical", length = 1, fun.name = function.name) ; eval(ee)**

**tempo <- fun\_check(data = graph.in.file, class = "vector", mode = "logical", length = 1, fun.name = function.name) ; eval(ee)**

**if(tempo$problem == FALSE & graph.in.file == TRUE & is.null(dev.list())){**

**tempo.cat <- paste0("ERROR IN ", function.name, ": \ngraph.in.file PARAMETER SET TO TRUE BUT NO ACTIVE GRAPHIC DEVICE DETECTED")**

**text.check <- c(text.check, tempo.cat)**

**arg.check <- c(arg.check, TRUE)**

**}else if(tempo$problem == FALSE & graph.in.file == TRUE & ! is.null(dev.list())){**

**warn.count <- warn.count + 1**

**tempo.warn <- paste0("(", warn.count,") GRAPHS PRINTED IN THE CURRENT DEVICE (TYPE ", toupper(names(dev.cur())), ")")**

**warn <- paste0(ifelse(is.null(warn), tempo.warn, paste0(warn, "\n\n", tempo.warn)))**

**}**

**if( ! is.null(lib.path)){**

**tempo <- fun\_check(data = lib.path, class = "vector", mode = "character", fun.name = function.name) ; eval(ee)**

**if(tempo$problem == FALSE){**

**if( ! all(dir.exists(lib.path))){** *# separation to avoid the problem of tempo$problem == FALSE and lib.path == NA*

**tempo.cat <- paste0("ERROR IN ", function.name, ": DIRECTORY PATH INDICATED IN THE lib.path ARGUMENT DOES NOT EXISTS:\n", paste(lib.path, collapse = "\n"))**

**text.check <- c(text.check, tempo.cat)**

**arg.check <- c(arg.check, TRUE)**

**}**

**}**

**}**

**}**

**if(any(arg.check) == TRUE){**

**stop(paste0("\n\n================\n\n", paste(text.check[arg.check], collapse = "\n"), "\n\n================\n\n", ifelse(is.null(warn), "", paste0("IN ADDITION\nWARNING", ifelse(warn.count > 1, "S", ""), ":\n\n", warn))), call. = FALSE)** *#*

**}**

*# source("C:/Users/Gael/Documents/Git\_versions\_to\_use/debugging\_tools\_for\_r\_dev-v1.2/r\_debugging\_tools-v1.2.R") ; eval(parse(text = str\_basic\_arg\_check\_dev)) ; eval(parse(text = str\_arg\_check\_with\_fun\_check\_dev)) # activate this line and use the function (with no arguments left as NULL) to check arguments status and if they have been checked using fun\_check()*

*# end argument checking*

*# other required function checking*

**if(plot == TRUE){**

**if(length(utils::find("fun\_pack", mode = "function")) == 0){**

**tempo.cat <- paste0("ERROR IN ", function.name, ": REQUIRED fun\_pack() FUNCTION IS MISSING IN THE R ENVIRONMENT")**

**stop(paste0("\n\n================\n\n", tempo.cat, "\n\n================\n\n", ifelse(is.null(warn), "", paste0("IN ADDITION\nWARNING", ifelse(warn.count > 1, "S", ""), ":\n\n", warn))), call. = FALSE)** *# == in stop() to be able to add several messages between ==*

**}**

**if(length(utils::find("fun\_open", mode = "function")) == 0){**

**tempo.cat <- paste0("ERROR IN ", function.name, ": REQUIRED fun\_open() FUNCTION IS MISSING IN THE R ENVIRONMENT")**

**stop(paste0("\n\n================\n\n", tempo.cat, "\n\n================\n\n", ifelse(is.null(warn), "", paste0("IN ADDITION\nWARNING", ifelse(warn.count > 1, "S", ""), ":\n\n", warn))), call. = FALSE)** *# == in stop() to be able to add several messages between ==*

**}**

**if(length(utils::find("fun\_gg\_palette", mode = "function")) == 0){**

**tempo.cat <- paste0("ERROR IN ", function.name, ": REQUIRED fun\_gg\_palette() FUNCTION IS MISSING IN THE R ENVIRONMENT")**

**stop(paste0("\n\n================\n\n", tempo.cat, "\n\n================\n\n", ifelse(is.null(warn), "", paste0("IN ADDITION\nWARNING", ifelse(warn.count > 1, "S", ""), ":\n\n", warn))), call. = FALSE)** *# == in stop() to be able to add several messages between ==*

**}**

**if(length(utils::find("fun\_gg\_empty\_graph", mode = "function")) == 0){**

**tempo.cat <- paste0("ERROR IN ", function.name, ": REQUIRED fun\_gg\_empty\_graph() FUNCTION IS MISSING IN THE R ENVIRONMENT")**

**stop(paste0("\n\n================\n\n", tempo.cat, "\n\n================\n\n", ifelse(is.null(warn), "", paste0("IN ADDITION\nWARNING", ifelse(warn.count > 1, "S", ""), ":\n\n", warn))), call. = FALSE)** *# == in stop() to be able to add several messages between ==*

**}**

**if(length(utils::find("fun\_gg\_scatter", mode = "function")) == 0){**

**tempo.cat <- paste0("ERROR IN ", function.name, ": REQUIRED fun\_gg\_scatter() FUNCTION IS MISSING IN THE R ENVIRONMENT")**

**stop(paste0("\n\n================\n\n", tempo.cat, "\n\n================\n\n", ifelse(is.null(warn), "", paste0("IN ADDITION\nWARNING", ifelse(warn.count > 1, "S", ""), ":\n\n", warn))), call. = FALSE)** *# == in stop() to be able to add several messages between ==*

**}**

**if(length(utils::find("fun\_close", mode = "function")) == 0){**

**tempo.cat <- paste0("ERROR IN ", function.name, ": REQUIRED fun\_close() FUNCTION IS MISSING IN THE R ENVIRONMENT")**

**stop(paste0("\n\n================\n\n", tempo.cat, "\n\n================\n\n", ifelse(is.null(warn), "", paste0("IN ADDITION\nWARNING", ifelse(warn.count > 1, "S", ""), ":\n\n", warn))), call. = FALSE)** *# == in stop() to be able to add several messages between ==*

**}**

**}**

*# end other required function checking*

*# package checking*

**if(plot == TRUE){**

**fun\_pack(req.package = c("ggplot2"), lib.path = lib.path)**

**}**

*# end package checking*

*# main code*

*# na and Inf detection and removal (done now to be sure of the correct length of categ)*

**data1.removed.row.nb <- NULL**

**data1.removed.rows <- NULL**

**data2.removed.row.nb <- NULL**

**data2.removed.rows <- NULL**

**if(any(is.na(data1[, c(x1, y1)])) | any(is.infinite(data1[, x1])) | any(is.infinite(data1[, y1]))){**

**tempo.na <- unlist(lapply(lapply(c(data1[c(x1, y1)]), FUN = is.na), FUN = which))**

**tempo.inf <- unlist(lapply(lapply(c(data1[c(x1, y1)]), FUN = is.infinite), FUN = which))**

**data1.removed.row.nb <- sort(unique(c(tempo.na, tempo.inf)))**

**if(length(data1.removed.row.nb) > 0){**

**data1.removed.rows <- data1[data1.removed.row.nb, ]**

**}**

**if(length(data1.removed.row.nb) == nrow(data1)){**

**tempo.cat <- paste0("ERROR IN ", function.name, ": AT LEAST ONE NA, NaN, -Inf OR Inf DETECTED IN EACH ROW OF data1. FUNCTION CANNOT BE USED ON EMPTY DATA FRAME")**

**stop(paste0("\n\n================\n\n", tempo.cat, "\n\n================\n\n", ifelse(is.null(warn), "", paste0("IN ADDITION\nWARNING", ifelse(warn.count > 1, "S", ""), ":\n\n", warn))), call. = FALSE)** *# == in stop() to be able to add several messages between ==*

**}**

**if(length(data1.removed.row.nb) > 0){**

**data1 <- data1[-data1.removed.row.nb, ]**

**}**

**if(nrow(data1) == 0){**

**tempo.cat <- paste0("ERROR IN ", function.name, ": CODE INCONSISTENCY 1")**

**stop(paste0("\n\n================\n\n", tempo.cat, "\n\n================\n\n", ifelse(is.null(warn), "", paste0("IN ADDITION\nWARNING", ifelse(warn.count > 1, "S", ""), ":\n\n", warn))), call. = FALSE)** *# == in stop() to be able to add several messages between ==*

**}**

**warn.count <- warn.count + 1**

**tempo.warn <- paste0("(", warn.count,") NA, NaN, -Inf OR Inf DETECTED IN COLUMN ", paste(c(x1, y1), collapse = " "), " OF data1 AND CORRESPONDING ROWS REMOVED (SEE $data1.removed.row.nb AND $data1.removed.rows)")**

**warn <- paste0(ifelse(is.null(warn), tempo.warn, paste0(warn, "\n\n", tempo.warn)))**

**}else{**

**warn.count <- warn.count + 1**

**tempo.warn <- paste0("(", warn.count,") NO NA, NaN, -Inf OR Inf DETECTED IN COLUMN ", paste(c(x1, y1), collapse = " "), " OF data1. NO ROW REMOVED")**

**warn <- paste0(ifelse(is.null(warn), tempo.warn, paste0(warn, "\n\n", tempo.warn)))**

**}**

**if( ! is.null(data2)){**

**if(any(is.na(data2[, c(x2, y2)])) | any(is.infinite(data2[, x2])) | any(is.infinite(data2[, y2]))){**

**tempo.na <- unlist(lapply(lapply(c(data2[c(x2, y2)]), FUN = is.na), FUN = which))**

**tempo.inf <- unlist(lapply(lapply(c(data2[c(x2, y2)]), FUN = is.infinite), FUN = which))**

**data2.removed.row.nb <- sort(unique(c(tempo.na, tempo.inf)))**

**if(length(data2.removed.row.nb) > 0){**

**data2.removed.rows <- data2[data2.removed.row.nb, ]**

**}**

**if(length(data2.removed.row.nb) == nrow(data2)){**

**tempo.cat <- paste0("ERROR IN ", function.name, ": AT LEAST ONE NA, NaN, -Inf OR Inf DETECTED IN EACH ROW OF data2. FUNCTION CANNOT BE USED ON EMPTY DATA FRAME")**

**stop(paste0("\n\n================\n\n", tempo.cat, "\n\n================\n\n", ifelse(is.null(warn), "", paste0("IN ADDITION\nWARNING", ifelse(warn.count > 1, "S", ""), ":\n\n", warn))), call. = FALSE)** *# == in stop() to be able to add several messages between ==*

**}**

**if(length(data2.removed.row.nb) > 0){**

**data2 <- data2[-data2.removed.row.nb, ]**

**}**

**if(nrow(data2) == 0){**

**tempo.cat <- paste0("ERROR IN ", function.name, ": CODE INCONSISTENCY 2")**

**stop(paste0("\n\n================\n\n", tempo.cat, "\n\n================\n\n", ifelse(is.null(warn), "", paste0("IN ADDITION\nWARNING", ifelse(warn.count > 1, "S", ""), ":\n\n", warn))), call. = FALSE)** *# == in stop() to be able to add several messages between ==*

**}**

**warn.count <- warn.count + 1**

**tempo.warn <- paste0("(", warn.count,") NA, NaN, -Inf OR Inf DETECTED IN COLUMN ", paste(c(x2, y2), collapse = " "), " OF data2 AND CORRESPONDING ROWS REMOVED (SEE $data2.removed.row.nb AND $data2.removed.rows)")**

**warn <- paste0(ifelse(is.null(warn), tempo.warn, paste0(warn, "\n\n", tempo.warn)))**

**}else{**

**warn.count <- warn.count + 1**

**tempo.warn <- paste0("(", warn.count,") NO NA, NaN, -Inf OR Inf DETECTED IN COLUMN ", paste(c(x2, y2), collapse = " "), " OF data2. NO ROW REMOVED")**

**warn <- paste0(ifelse(is.null(warn), tempo.warn, paste0(warn, "\n\n", tempo.warn)))**

**print(warn)**

**}**

**}**

*# end na and Inf detection and removal (done now to be sure of the correct length of categ)*

*# row annotation (dot number)*

*# data1 <- data1[ ! duplicated(data1[, c(x1, y1)]), ] # do not remove the dots that have same x and y values, because they will have different dot number -> not the same position on the matrices (so true for symmetric matrices)*

**data1 <- cbind(data1, DOT\_NB = 1:nrow(data1))**

**if( ! is.null(data2)){**

*# data2 <- data2[ ! duplicated(data2[, c(x2, y2)]), ] # do not remove the dots that have same x and y values, because they will have different dot number -> not the same position on the matrices (so true for symmetric matrices)*

**data2 <- cbind(data2, DOT\_NB = 1:nrow(data2))**

**}**

*# end row annotation (dot number)*

*# Method using x unit interval*

*# may be create vector of each column to increase speed*

**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*

**if(any(data1[, x1] %in% c(Inf, -Inf))){**

**warn.count <- warn.count + 1**

**tempo.warn <- paste0("(", warn.count,") THE data1 ARGUMENT CONTAINS -Inf OR Inf VALUES IN THE x1 COLUMN, THAT WILL NOT BE CONSIDERED IN THE PLOT RANGE")**

**warn <- paste0(ifelse(is.null(warn), tempo.warn, paste0(warn, "\n\n", tempo.warn)))**

**}**

**x.range <- range(data1[, x1], na.rm = TRUE, finite = TRUE)** *# finite = TRUE removes all the -Inf and Inf except if only this. In that case, whatever the -Inf and/or Inf present, output -Inf;Inf range. Idem with NA only*

**if(suppressWarnings(any(x.range %in% c(Inf, -Inf)))){**

**tempo.cat <- paste0("ERROR IN ", function.name, " COMPUTED x.range CONTAINS Inf VALUES, BECAUSE VALUES FROM data1 ARGUMENTS ARE NA OR Inf ONLY")**

**stop(paste0("\n\n================\n\n", tempo.cat, "\n\n================\n\n", ifelse(is.null(warn), "", paste0("IN ADDITION\nWARNING", ifelse(warn.count > 1, "S", ""), ":\n\n", warn))), call. = FALSE)** *# == in stop() to be able to add several messages between ==*

**}**

**if(any(data1[, y1] %in% c(Inf, -Inf))){**

**warn.count <- warn.count + 1**

**tempo.warn <- paste0("(", warn.count,") THE data1 ARGUMENT CONTAINS -Inf OR Inf VALUES IN THE y1 COLUMN, THAT WILL NOT BE CONSIDERED IN THE PLOT RANGE")**

**warn <- paste0(ifelse(is.null(warn), tempo.warn, paste0(warn, "\n\n", tempo.warn)))**

**}**

**y.range <- range(data1[, y1], na.rm = TRUE, finite = TRUE)** *# finite = TRUE removes all the -Inf and Inf except if only this. In that case, whatever the -Inf and/or Inf present, output -Inf;Inf range. Idem with NA only*

**if(suppressWarnings(any(x.range %in% c(Inf, -Inf)))){**

**tempo.cat <- paste0("ERROR IN ", function.name, " COMPUTED y.range CONTAINS Inf VALUES, BECAUSE VALUES FROM data1 ARGUMENTS ARE NA OR Inf ONLY")**

**stop(paste0("\n\n================\n\n", tempo.cat, "\n\n================\n\n", ifelse(is.null(warn), "", paste0("IN ADDITION\nWARNING", ifelse(warn.count > 1, "S", ""), ":\n\n", warn))), call. = FALSE)** *# == in stop() to be able to add several messages between ==*

**}**

**x.range.plot <- range(data1[, x1], na.rm = TRUE, finite = TRUE)** *# finite = TRUE removes all the -Inf and Inf except if only this. In that case, whatever the -Inf and/or Inf present, output -Inf;Inf range. Idem with NA only*

**y.range.plot <- range(data1[, y1], na.rm = TRUE, finite = TRUE)** *# finite = TRUE removes all the -Inf and Inf except if only this. In that case, whatever the -Inf and/or Inf present, output -Inf;Inf range. Idem with NA only*

**if( ! is.null(data2)){**

**if(any(data2[, x2] %in% c(Inf, -Inf))){**

**warn.count <- warn.count + 1**

**tempo.warn <- paste0("(", warn.count,") THE data2 ARGUMENT CONTAINS -Inf OR Inf VALUES IN THE x2 COLUMN, THAT WILL NOT BE CONSIDERED IN THE PLOT RANGE")**

**warn <- paste0(ifelse(is.null(warn), tempo.warn, paste0(warn, "\n\n", tempo.warn)))**

**}**

**x.range.plot <- range(data1[, x1], data2[, x2], na.rm = TRUE, finite = TRUE)** *# finite = TRUE removes all the -Inf and Inf except if only this. In that case, whatever the -Inf and/or Inf present, output -Inf;Inf range. Idem with NA only*

**if(any(data2[, y2] %in% c(Inf, -Inf))){**

**warn.count <- warn.count + 1**

**tempo.warn <- paste0("(", warn.count,") THE data2 ARGUMENT CONTAINS -Inf OR Inf VALUES IN THE y2 COLUMN, THAT WILL NOT BE CONSIDERED IN THE PLOT RANGE")**

**warn <- paste0(ifelse(is.null(warn), tempo.warn, paste0(warn, "\n\n", tempo.warn)))**

**}**

**y.range.plot <- range(data1[, y1], data2[, y2], na.rm = TRUE, finite = TRUE)** *# finite = TRUE removes all the -Inf and Inf except if only this. In that case, whatever the -Inf and/or Inf present, output -Inf;Inf range. Idem with NA only*

**}**

**if(suppressWarnings(any(x.range.plot %in% c(Inf, -Inf)))){**

**tempo.cat <- paste0("ERROR IN ", function.name, " COMPUTED x.range.plot CONTAINS Inf VALUES, BECAUSE VALUES FROM data1 (AND data2?) ARGUMENTS ARE NA OR Inf ONLY")**

**stop(paste0("\n\n================\n\n", tempo.cat, "\n\n================\n\n", ifelse(is.null(warn), "", paste0("IN ADDITION\nWARNING", ifelse(warn.count > 1, "S", ""), ":\n\n", warn))), call. = FALSE)** *# == in stop() to be able to add several messages between ==*

**}**

**if(suppressWarnings(any(y.range.plot %in% c(Inf, -Inf)))){**

**tempo.cat <- paste0("ERROR IN ", function.name, " COMPUTED y.range.plot CONTAINS Inf VALUES, BECAUSE VALUES FROM data1 (AND data2?) ARGUMENTS ARE NA OR Inf ONLY")**

**stop(paste0("\n\n================\n\n", tempo.cat, "\n\n================\n\n", ifelse(is.null(warn), "", paste0("IN ADDITION\nWARNING", ifelse(warn.count > 1, "S", ""), ":\n\n", warn))), call. = FALSE)** *# == in stop() to be able to add several messages between ==*

**}**

**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*

**y.unknown.data2.dot.nb <- integer()** *# vector that will contain the 1D coordinates (i.e., dots) of data2 that are problematic: data2 dots outside of the range of data1, or data2 dots in a sliding window without data1 dots*

*# recover data2 dots outside the range of data1*

**if(any(data2[, x2] < x.range[1])){**

**y.unknown.data2.dot.nb <- c(y.unknown.data2.dot.nb, data2$DOT\_NB[data2[, x2] < x.range[1]])**

**#tempo.warn & indicate the interval**

**}**

**if(any(data2[, x2] > x.range[2])){**

**y.unknown.data2.dot.nb <- c(y.unknown.data2.dot.nb, data2$DOT\_NB[data2[, x2] > x.range[2]])**

**#tempo.warn & indicate the interval**

**}**

*# 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 window but no data1 dots to generates the quantiles*

**y.unknown.data2.dot.nb <- c(y.unknown.data2.dot.nb, data2$DOT\_NB[x.data2.dot.here])**

**y.unknown.data2.dot.nb <- unique(y.unknown.data2.dot.nb)**

**#tempo.warn & indicate the interval**

**# tempo.warn <- paste0("FROM FUNCTION ", function.name, ": 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 ROW NUMBERS ARE:\n", paste(which(x.data1.dot.here == FALSE & x.data2.dot.here == TRUE), collapse = "\n"))**

**# warn <- paste0(ifelse(is.null(warn), tempo.warn, paste0(warn, "\n\n", tempo.warn)))**

**}**

**}**

*# 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("ERROR IN ", function.name, ": THE SLIDING WINDOW HAS NOT REACHED THE MAX VALUE OF data1 ON THE X-AXIS: ", max.pos, " VERSUS ", x.range[2])**

**stop(paste0("\n\n================\n\n", tempo.cat, "\n\n================\n\n", ifelse(is.null(warn), "", paste0("IN ADDITION\nWARNING", ifelse(warn.count > 1, "S", ""), ":\n\n", warn))), call. = FALSE)** *# == in stop() to be able to add several messages between ==*

**}**

**y.incon.data1.dot.nb.final <- unique(c(y.outside.data1.dot.nb[y.outside.data1.dot.nb %in% y.inside.data1.dot.nb], y.inside.data1.dot.nb[y.inside.data1.dot.nb %in% y.outside.data1.dot.nb]))** *# inconsistent dots: if a row number of y.inside.data1.dot.nb is present in y.outside.data1.dot.nb (and vice versa), it means that during the sliding, a dot has been sometime inside, sometime outside -> removed from the outside list*

**y.outside.data1.dot.nb.final <- y.outside.data1.dot.nb[ ! (y.outside.data1.dot.nb %in% y.incon.data1.dot.nb.final)]** *# inconsistent dots removed from the outside list*

**y.inside.data1.dot.nb.final <- y.inside.data1.dot.nb[ ! (y.inside.data1.dot.nb %in% y.incon.data1.dot.nb.final)]** *# inconsistent dots removed from the inside list*

**if( ! is.null(data2)){**

*# if some unknown dots are also inside, and/or outside, they are put in the inside and/or outside. Ok, because then the intersection between inside and outside is treated -> inconsistent dots*

**tempo.unknown.out <- y.unknown.data2.dot.nb[y.unknown.data2.dot.nb %in% y.outside.data2.dot.nb]**

**y.outside.data2.dot.nb <- unique(c(y.outside.data2.dot.nb, tempo.unknown.out))** *# if a row number of y.unknown.data2.dot.nb is present in y.outside.data2.dot.nb, it is put into outside*

**tempo.unknown.in <- y.unknown.data2.dot.nb[y.unknown.data2.dot.nb %in% y.inside.data2.dot.nb]**

**y.inside.data2.dot.nb <- unique(c(y.inside.data2.dot.nb, tempo.unknown.in))** *# if a row number of y.unknown.data2.dot.nb is present in y.inside.data2.dot.nb, it is put into inside*

**y.unknown.data2.dot.nb.final <- y.unknown.data2.dot.nb[ ! (y.unknown.data2.dot.nb %in% c(y.outside.data2.dot.nb, y.inside.data2.dot.nb))]** *# then dots also in inside and outside are remove from unknown*

**y.incon.data2.dot.nb.final <- unique(c(y.outside.data2.dot.nb[y.outside.data2.dot.nb %in% y.inside.data2.dot.nb], y.inside.data2.dot.nb[y.inside.data2.dot.nb %in% y.outside.data2.dot.nb]))** *# inconsistent dots: if a row number of y.inside.data2.dot.nb is present in y.outside.data2.dot.nb (and vice versa), it means that during the sliding, a dot has been sometime inside, sometime outside -> removed from the outside list*

**y.outside.data2.dot.nb.final <- y.outside.data2.dot.nb[ ! (y.outside.data2.dot.nb %in% y.incon.data2.dot.nb.final)]** *# inconsistent dots removed from the outside list*

**y.inside.data2.dot.nb.final <- y.inside.data2.dot.nb[ ! (y.inside.data2.dot.nb %in% y.incon.data2.dot.nb.final)]** *# inconsistent dots removed from the inside list*

**}**

*# 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 window 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*

**x.unknown.data2.dot.nb <- integer()** *# vector that will contain the 1D coordinates (i.e., dots) of data2 that are problematic: data2 dots outside of the range of data1, or data2 dots in a sliding window without data1 dots*

*# recover data2 dots outside the range of data1*

**if(any(data2[, y2] < y.range[1])){**

**x.unknown.data2.dot.nb <- c(x.unknown.data2.dot.nb, data2$DOT\_NB[data2[, y2] < y.range[1]])**

**}**

**if(any(data2[, y2] > y.range[2])){**

**x.unknown.data2.dot.nb <- c(x.unknown.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*

**x.unknown.data2.dot.nb <- c(x.unknown.data2.dot.nb, data2$DOT\_NB[y.data2.dot.here])**

**x.unknown.data2.dot.nb <- unique(x.unknown.data2.dot.nb)**

**# tempo.warn <- paste0("FROM FUNCTION ", function.name, ": 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 ROW NUMBERS ARE:\n", paste(which(y.data1.dot.here == FALSE & y.data2.dot.here == TRUE), collapse = "\n"))**

**# warn <- paste0(ifelse(is.null(warn), tempo.warn, paste0(warn, "\n\n", tempo.warn)))**

**}**

**}**

*# 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("ERROR IN ", function.name, ": THE SLIDING WINDOW HAS NOT REACHED THE MAX VALUE OF data1 ON THE Y-AXIS: ", max.pos, " VERSUS ", y.range[2])**

**stop(paste0("\n\n================\n\n", tempo.cat, "\n\n================\n\n", ifelse(is.null(warn), "", paste0("IN ADDITION\nWARNING", ifelse(warn.count > 1, "S", ""), ":\n\n", warn))), call. = FALSE)** *# == in stop() to be able to add several messages between ==*

**}**

**x.incon.data1.dot.nb.final <- unique(c(x.outside.data1.dot.nb[x.outside.data1.dot.nb %in% x.inside.data1.dot.nb], x.inside.data1.dot.nb[x.inside.data1.dot.nb %in% x.outside.data1.dot.nb]))** *# inconsistent dots: if a row number of x.inside.data1.dot.nb is present in x.outside.data1.dot.nb (and vice versa), it means that during the sliding, a dot has been sometime inside, sometime outside -> removed from the outside list*

**x.outside.data1.dot.nb.final <- x.outside.data1.dot.nb[ ! (x.outside.data1.dot.nb %in% x.incon.data1.dot.nb.final)]** *# inconsistent dots removed from the outside list*

**x.inside.data1.dot.nb.final <- x.inside.data1.dot.nb[ ! (x.inside.data1.dot.nb %in% x.incon.data1.dot.nb.final)]** *# inconsistent dots removed from the inside list*

**if( ! is.null(data2)){**

*# if some unknown dots are also inside, and/or outside, they are put in the inside and/or outside. Ok, because then the intersection between inside and outside is treated -> inconsistent dots*

**tempo.unknown.out <- x.unknown.data2.dot.nb[x.unknown.data2.dot.nb %in% x.outside.data2.dot.nb]**

**x.outside.data2.dot.nb <- unique(c(x.outside.data2.dot.nb, tempo.unknown.out))** *# if a row number of x.unknown.data2.dot.nb is present in x.outside.data2.dot.nb, it is put into outside*

**tempo.unknown.in <- x.unknown.data2.dot.nb[x.unknown.data2.dot.nb %in% x.inside.data2.dot.nb]**

**x.inside.data2.dot.nb <- unique(c(x.inside.data2.dot.nb, tempo.unknown.in))** *# if a row number of x.unknown.data2.dot.nb is present in x.inside.data2.dot.nb, it is put into inside*

**x.unknown.data2.dot.nb.final <- x.unknown.data2.dot.nb[ ! (x.unknown.data2.dot.nb %in% c(x.outside.data2.dot.nb, x.inside.data2.dot.nb))]** *# then dots also in inside and outside are remove from unknown*

**x.incon.data2.dot.nb.final <- unique(c(x.outside.data2.dot.nb[x.outside.data2.dot.nb %in% x.inside.data2.dot.nb], x.inside.data2.dot.nb[x.inside.data2.dot.nb %in% x.outside.data2.dot.nb]))** *# inconsistent dots: if a row number of x.inside.data2.dot.nb is present in x.outside.data2.dot.nb (and vice versa), it means that during the sliding, a dot has been sometime inside, sometime outside -> removed from the outside list*

**x.outside.data2.dot.nb.final <- x.outside.data2.dot.nb[ ! (x.outside.data2.dot.nb %in% x.incon.data2.dot.nb.final)]** *# inconsistent dots removed from the outside list*

**x.inside.data2.dot.nb.final <- x.inside.data2.dot.nb[ ! (x.inside.data2.dot.nb %in% x.incon.data2.dot.nb.final)]** *# inconsistent dots removed from the inside list*

**}**

*# end y-axis sliding and x-axis limits of the data1 cloud -> x significant data2*

**}**

*# end Method using y unit interval*

*# recovering the frame coordinates*

**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"}**

**)**

**)**

*# end recovering the frame coordinates*

*# recovering the dot coordinates*

**data1.signif.dot <- NULL**

**data1.non.signif.dot <- NULL**

**data1.incon.dot <- NULL**

**data2.signif.dot <- NULL**

**data2.non.signif.dot <- NULL**

**data2.unknown.dot <- NULL**

**data2.incon.dot <- NULL**

**if(( ! is.null(x.range.split)) & ( ! is.null(y.range.split))){**

*# inconsistent dots recovery*

**if(length(unique(c(x.incon.data1.dot.nb.final, y.incon.data1.dot.nb.final))) > 0){**

**data1.incon.dot <- data1[data1$DOT\_NB %in% unique(c(x.incon.data1.dot.nb.final, y.incon.data1.dot.nb.final)), ]** *# if a dot in inconsistent in x or y -> classified as inconsistent (so unique() used)*

*# removal of the inconsistent dot in the other classifications*

**x.inside.data1.dot.nb.final <- x.inside.data1.dot.nb.final[ ! x.inside.data1.dot.nb.final %in% data1.incon.dot$DOT\_NB]**

**y.inside.data1.dot.nb.final <- y.inside.data1.dot.nb.final[ ! y.inside.data1.dot.nb.final %in% data1.incon.dot$DOT\_NB]**

**x.outside.data1.dot.nb.final <- x.outside.data1.dot.nb.final[ ! x.outside.data1.dot.nb.final %in% data1.incon.dot$DOT\_NB]**

**y.outside.data1.dot.nb.final <- y.outside.data1.dot.nb.final[ ! y.outside.data1.dot.nb.final %in% data1.incon.dot$DOT\_NB]**

**x.unknown.data1.dot.nb.final <- x.unknown.data1.dot.nb.final[ ! x.unknown.data1.dot.nb.final %in% data1.incon.dot$DOT\_NB]**

**y.unknown.data1.dot.nb.final <- y.unknown.data1.dot.nb.final[ ! y.unknown.data1.dot.nb.final %in% data1.incon.dot$DOT\_NB]**

*# end removal of the inconsistent dot in the other classifications*

**}**

**if( ! is.null(data2)){**

**if(length(unique(c(x.incon.data2.dot.nb.final, y.incon.data2.dot.nb.final))) > 0){**

**data2.incon.dot <- data2[data2$DOT\_NB %in% unique(c(x.incon.data2.dot.nb.final, y.incon.data2.dot.nb.final)), ]**

*# removal of the inconsistent dot in the other classifications*

**x.inside.data2.dot.nb.final <- x.inside.data2.dot.nb.final[ ! x.inside.data2.dot.nb.final %in% data2.incon.dot$DOT\_NB]**

**y.inside.data2.dot.nb.final <- y.inside.data2.dot.nb.final[ ! y.inside.data2.dot.nb.final %in% data2.incon.dot$DOT\_NB]**

**x.outside.data2.dot.nb.final <- x.outside.data2.dot.nb.final[ ! x.outside.data2.dot.nb.final %in% data2.incon.dot$DOT\_NB]**

**y.outside.data2.dot.nb.final <- y.outside.data2.dot.nb.final[ ! y.outside.data2.dot.nb.final %in% data2.incon.dot$DOT\_NB]**

**x.unknown.data2.dot.nb.final <- x.unknown.data2.dot.nb.final[ ! x.unknown.data2.dot.nb.final %in% data2.incon.dot$DOT\_NB]**

**y.unknown.data2.dot.nb.final <- y.unknown.data2.dot.nb.final[ ! y.unknown.data2.dot.nb.final %in% data2.incon.dot$DOT\_NB]**

*# end removal of the inconsistent dot in the other classifications*

**}**

**}**

*# end inconsistent dots recovery*

*# unknown dots recovery*

**if( ! is.null(data2)){**

**if(data2.pb.dot == "signif"){**

**x.outside.data2.dot.nb.final <- unique(c(x.outside.data2.dot.nb.final, x.unknown.data2.dot.nb.final))**

**x.inside.data2.dot.nb.final <- x.inside.data2.dot.nb.final[ ! x.inside.data2.dot.nb.final %in% x.unknown.data2.dot.nb.final]** *# remove x.unknown.data2.dot.nb.final from x.inside.data2.dot.nb.final*

**y.outside.data2.dot.nb.final <- unique(c(y.outside.data2.dot.nb.final, y.unknown.data2.dot.nb.final))**

**y.inside.data2.dot.nb.final <- y.inside.data2.dot.nb.final[ ! y.inside.data2.dot.nb.final %in% y.unknown.data2.dot.nb.final]** *# remove y.unknown.data2.dot.nb.final from y.inside.data2.dot.nb.final*

**x.unknown.data2.dot.nb.final <- NULL**

**y.unknown.data2.dot.nb.final <- NULL**

**data2.unknown.dot <- NULL**

**}else if(data2.pb.dot == "not.signif"){**

**x.inside.data2.dot.nb.final <- unique(c(x.inside.data2.dot.nb.final, x.unknown.data2.dot.nb.final))**

**x.outside.data2.dot.nb.final <- x.outside.data2.dot.nb.final[ ! x.outside.data2.dot.nb.final %in% x.unknown.data2.dot.nb.final]** *# remove x.unknown.data2.dot.nb.final from x.outside.data2.dot.nb.final*

**y.inside.data2.dot.nb.final <- unique(c(y.inside.data2.dot.nb.final, y.unknown.data2.dot.nb.final))**

**y.outside.data2.dot.nb.final <- y.outside.data2.dot.nb.final[ ! y.outside.data2.dot.nb.final %in% y.unknown.data2.dot.nb.final]** *# remove y.unknown.data2.dot.nb.final from y.outside.data2.dot.nb.final*

**x.unknown.data2.dot.nb.final <- NULL**

**y.unknown.data2.dot.nb.final <- NULL**

**data2.unknown.dot <- NULL**

**}else if(data2.pb.dot == "unknown"){**

**if(length(unique(c(x.unknown.data2.dot.nb.final, y.unknown.data2.dot.nb.final))) > 0){**

**data2.unknown.dot <- data2[data2$DOT\_NB %in% unique(c(x.unknown.data2.dot.nb.final, y.unknown.data2.dot.nb.final)), ]** *# if a dot in unknown in x or y -> classified as unknown (so unique() used)*

**x.outside.data2.dot.nb.final <- x.outside.data2.dot.nb.final[ ! x.outside.data2.dot.nb.final %in% data2.unknown.dot$DOT\_NB]** *# remove x.unknown.data2.dot.nb.final from x.outside.data2.dot.nb.final*

**x.inside.data2.dot.nb.final <- x.inside.data2.dot.nb.final[ ! x.inside.data2.dot.nb.final %in% data2.unknown.dot$DOT\_NB]** *# remove x.unknown.data2.dot.nb.final from x.inside.data2.dot.nb.final*

**y.outside.data2.dot.nb.final <- y.outside.data2.dot.nb.final[ ! y.outside.data2.dot.nb.final %in% data2.unknown.dot$DOT\_NB]** *# remove y.unknown.data2.dot.nb.final from y.outside.data2.dot.nb.final*

**y.inside.data2.dot.nb.final <- y.inside.data2.dot.nb.final[ ! y.inside.data2.dot.nb.final %in% data2.unknown.dot$DOT\_NB]** *# remove y.unknown.data2.dot.nb.final from y.inside.data2.dot.nb.final*

**}**

**}else{**

**tempo.cat <- paste0("ERROR IN ", function.name, ": CODE INCONSISTENCY 3")**

**stop(paste0("\n\n================\n\n", tempo.cat, "\n\n================\n\n", ifelse(is.null(warn), "", paste0("IN ADDITION\nWARNING", ifelse(warn.count > 1, "S", ""), ":\n\n", warn))), call. = FALSE)** *# == in stop() to be able to add several messages between ==*

**}**

**}**

*# end unknown dots recovery*

*# sign and non sign dot recovery*

**if(xy.cross.kind == "|"){** *# here the problem is to deal with significant dots depending on x and y. Thus I start with that, recover dots finally non significant in outside and put them in inside (when &), and remove from inside the dots in outside*

**if(length(unique(c(x.outside.data1.dot.nb.final, y.outside.data1.dot.nb.final))) > 0){**

**tempo.outside <- unique(c(x.outside.data1.dot.nb.final, y.outside.data1.dot.nb.final))** *# union so unique() used*

**tempo.inside <- unique(c(x.inside.data1.dot.nb.final, y.inside.data1.dot.nb.final))**

**tempo.inside <- tempo.inside[ ! tempo.inside %in% tempo.outside]**

**data1.signif.dot <- data1[data1$DOT\_NB %in% tempo.outside, ]**

**data1.non.signif.dot <- data1[data1$DOT\_NB %in% tempo.inside, ]**

**}else{**

**data1.non.signif.dot <- data1[unique(c(x.inside.data1.dot.nb.final, y.inside.data1.dot.nb.final)), ]** *# if no outside dots, I recover all the inside dots and that's it*

**}**

**}else if(xy.cross.kind == "&"){**

**if(sum(x.outside.data1.dot.nb.final %in% y.outside.data1.dot.nb.final) > 0){** *# that is intersection*

**tempo.outside <- unique(x.outside.data1.dot.nb.final[x.outside.data1.dot.nb.final %in% y.outside.data1.dot.nb.final])** *# intersection*

**tempo.outside.removed <- unique(c(x.outside.data1.dot.nb.final, y.outside.data1.dot.nb.final))[ ! unique(c(x.outside.data1.dot.nb.final, y.outside.data1.dot.nb.final)) %in% tempo.outside]**

**tempo.inside <- unique(c(x.inside.data1.dot.nb.final, y.inside.data1.dot.nb.final))**

**data1.signif.dot <- data1[data1$DOT\_NB %in% tempo.outside, ]**

**data1.non.signif.dot <- data1[data1$DOT\_NB %in% tempo.inside, ]**

**}else{**

**data1.non.signif.dot <- data1[unique(c(x.inside.data1.dot.nb.final, y.inside.data1.dot.nb.final)), ]** *# if no outside dots, I recover all the inside dots and that's it*

**}**

**}else{**

**tempo.cat <- paste0("ERROR IN ", function.name, ": CODE INCONSISTENCY 4")**

**stop(paste0("\n\n================\n\n", tempo.cat, "\n\n================\n\n", ifelse(is.null(warn), "", paste0("IN ADDITION\nWARNING", ifelse(warn.count > 1, "S", ""), ":\n\n", warn))), call. = FALSE)** *# == in stop() to be able to add several messages between ==*

**}**

**if( ! is.null(data2)){**

**if(xy.cross.kind == "|"){** *# here the problem is to deal with significant dots depending on x and y. Thus I start with that, recover dots finally non significant in outside and put them in inside (when &), and remove from inside the dots in outside*

**if(length(unique(c(x.outside.data2.dot.nb.final, y.outside.data2.dot.nb.final))) > 0){**

**tempo.outside <- unique(c(x.outside.data2.dot.nb.final, y.outside.data2.dot.nb.final))** *# union so unique() used*

**tempo.inside <- unique(c(x.inside.data2.dot.nb.final, y.inside.data2.dot.nb.final))**

**tempo.inside <- tempo.inside[ ! tempo.inside %in% tempo.outside]**

**data2.signif.dot <- data2[data2$DOT\_NB %in% tempo.outside, ]**

**data2.non.signif.dot <- data2[data2$DOT\_NB %in% tempo.inside, ]**

**}else{**

**data2.non.signif.dot <- data2[unique(c(x.inside.data2.dot.nb.final, y.inside.data2.dot.nb.final)), ]** *# if no outside dots, I recover all the inside dots and that's it*

**}**

**}else if(xy.cross.kind == "&"){**

**if(sum(x.outside.data2.dot.nb.final %in% y.outside.data2.dot.nb.final) > 0){** *# that is intersection*

**tempo.outside <- unique(x.outside.data2.dot.nb.final[x.outside.data2.dot.nb.final %in% y.outside.data2.dot.nb.final])** *# intersection*

**tempo.outside.removed <- unique(c(x.outside.data2.dot.nb.final, y.outside.data2.dot.nb.final))[ ! unique(c(x.outside.data2.dot.nb.final, y.outside.data2.dot.nb.final)) %in% tempo.outside]**

**tempo.inside <- unique(c(x.inside.data2.dot.nb.final, y.inside.data2.dot.nb.final))**

**data2.signif.dot <- data2[data2$DOT\_NB %in% tempo.outside, ]**

**data2.non.signif.dot <- data2[data2$DOT\_NB %in% tempo.inside, ]**

**}else{**

**data2.non.signif.dot <- data2[unique(c(x.inside.data2.dot.nb.final, y.inside.data2.dot.nb.final)), ]** *# if no outside dots, I recover all the inside dots and that's it*

**}**

**}else{**

**tempo.cat <- paste0("ERROR IN ", function.name, ": CODE INCONSISTENCY 5")**

**stop(paste0("\n\n================\n\n", tempo.cat, "\n\n================\n\n", ifelse(is.null(warn), "", paste0("IN ADDITION\nWARNING", ifelse(warn.count > 1, "S", ""), ":\n\n", warn))), call. = FALSE)** *# == in stop() to be able to add several messages between ==*

**}**

**}**

*# end sign and non sign dot recovery*

**}else if(( ! is.null(x.range.split)) & is.null(y.range.split)){**

*# inconsistent dots recovery*

**if(length(y.incon.data1.dot.nb.final) > 0){**

**data1.incon.dot <- data1[data1$DOT\_NB %in% y.incon.data1.dot.nb.final, ]**

**}**

**if( ! is.null(data2)){**

**if(length(y.incon.data2.dot.nb.final) > 0){**

**data2.incon.dot <- data2[data2$DOT\_NB %in% y.incon.data2.dot.nb.final, ]**

**}**

**}***# end inconsistent dots recovery*

*# unknown dots recovery*

**if( ! is.null(data2)){**

**if(data2.pb.dot == "signif"){**

**y.outside.data2.dot.nb.final <- unique(c(y.outside.data2.dot.nb.final, y.unknown.data2.dot.nb.final))**

**}else if(data2.pb.dot == "not.signif"){**

**y.inside.data2.dot.nb.final <- unique(c(y.inside.data2.dot.nb.final, y.unknown.data2.dot.nb.final))**

**}else if(data2.pb.dot == "unknown"){**

**if(length(y.unknown.data2.dot.nb.final) > 0){**

**data2.unknown.dot <- data2[data2$DOT\_NB %in% y.unknown.data2.dot.nb.final, ]**

**}**

**}else{**

**tempo.cat <- paste0("ERROR IN ", function.name, ": CODE INCONSISTENCY 6")**

**stop(paste0("\n\n================\n\n", tempo.cat, "\n\n================\n\n", ifelse(is.null(warn), "", paste0("IN ADDITION\nWARNING", ifelse(warn.count > 1, "S", ""), ":\n\n", warn))), call. = FALSE)** *# == in stop() to be able to add several messages between ==*

**}**

**}**

*# end unknown dots recovery*

*# sign and non sign dot recovery*

**if(length(y.outside.data1.dot.nb.final) > 0){**

**data1.signif.dot <- data1[data1$DOT\_NB %in% y.outside.data1.dot.nb.final, ]**

**}**

**if(length(y.inside.data1.dot.nb.final) > 0){**

**data1.non.signif.dot <- data1[data1$DOT\_NB %in% y.inside.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, ]**

**}**

**if(length(y.inside.data2.dot.nb.final) > 0){**

**data2.non.signif.dot <- data2[data2$DOT\_NB %in% y.inside.data2.dot.nb.final, ]**

**}**

**}**

*# end sign and non sign dot recovery*

**}else if(is.null(x.range.split) & ( ! is.null(y.range.split))){**

*# inconsistent dots recovery*

**if(length(x.incon.data1.dot.nb.final) > 0){**

**data1.incon.dot <- data1[data1$DOT\_NB %in% x.incon.data1.dot.nb.final, ]**

**}**

**if( ! is.null(data2)){**

**if(length(x.incon.data2.dot.nb.final) > 0){**

**data2.incon.dot <- data2[data2$DOT\_NB %in% x.incon.data2.dot.nb.final, ]**

**}**

**}***# end inconsistent dots recovery*

*# unknown dots recovery*

**if( ! is.null(data2)){**

**if(data2.pb.dot == "signif"){**

**x.outside.data2.dot.nb.final <- unique(c(x.outside.data2.dot.nb.final, x.unknown.data2.dot.nb.final))**

**}else if(data2.pb.dot == "not.signif"){**

**x.inside.data2.dot.nb.final <- unique(c(x.inside.data2.dot.nb.final, x.unknown.data2.dot.nb.final))**

**}else if(data2.pb.dot == "unknown"){**

**if(length(x.unknown.data2.dot.nb.final) > 0){**

**data2.unknown.dot <- data2[data2$DOT\_NB %in% x.unknown.data2.dot.nb.final, ]**

**}**

**}else{**

**tempo.cat <- paste0("ERROR IN ", function.name, ": CODE INCONSISTENCY 7")**

**stop(paste0("\n\n================\n\n", tempo.cat, "\n\n================\n\n", ifelse(is.null(warn), "", paste0("IN ADDITION\nWARNING", ifelse(warn.count > 1, "S", ""), ":\n\n", warn))), call. = FALSE)** *# == in stop() to be able to add several messages between ==*

**}**

**}**

*# end unknown dots recovery*

*# sign and non sign dot recovery*

**if(length(x.outside.data1.dot.nb.final) > 0){**

**data1.signif.dot <- data1[data1$DOT\_NB %in% x.outside.data1.dot.nb.final, ]**

**}**

**if(length(x.inside.data1.dot.nb.final) > 0){**

**data1.non.signif.dot <- data1[data1$DOT\_NB %in% x.inside.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(length(x.inside.data2.dot.nb.final) > 0){**

**data2.non.signif.dot <- data2[data2$DOT\_NB %in% x.inside.data2.dot.nb.final, ]**

**}**

**}**

*# end sign and non sign dot recovery*

**}**

*# end recovering the dot coordinates*

*# verif*

**if(any(data1.signif.dot$DOT\_NB %in% data1.non.signif.dot$DOT\_NB)){**

**tempo.cat <- paste0("ERROR IN ", FUNCTION.NAME, ": CODE INCONSISTENCY 8")**

**stop(paste0("\n\n================\n\n", tempo.cat, "\n\n================\n\n", ifelse(is.null(warn), "", paste0("IN ADDITION\nWARNING", ifelse(warn.count > 1, "S", ""), ":\n\n", warn))), call. = FALSE)** *# == in stop() to be able to add several messages between ==*

**}**

**if(any(data1.non.signif.dot$DOT\_NB %in% data1.signif.dot$DOT\_NB)){**

**tempo.cat <- paste0("ERROR IN ", FUNCTION.NAME, ": CODE INCONSISTENCY 9")**

**stop(paste0("\n\n================\n\n", tempo.cat, "\n\n================\n\n", ifelse(is.null(warn), "", paste0("IN ADDITION\nWARNING", ifelse(warn.count > 1, "S", ""), ":\n\n", warn))), call. = FALSE)** *# == in stop() to be able to add several messages between ==*

**}**

**if(any(data1.signif.dot$DOT\_NB %in% data1.incon.dot$DOT\_NB)){**

**tempo.cat <- paste0("ERROR IN ", function.name, ": CODE INCONSISTENCY 10")**

**stop(paste0("\n\n================\n\n", tempo.cat, "\n\n================\n\n", ifelse(is.null(warn), "", paste0("IN ADDITION\nWARNING", ifelse(warn.count > 1, "S", ""), ":\n\n", warn))), call. = FALSE)** *# == in stop() to be able to add several messages between ==*

**}**

**if(any(data1.incon.dot$DOT\_NB %in% data1.signif.dot$DOT\_NB)){**

**tempo.cat <- paste0("ERROR IN ", function.name, ": CODE INCONSISTENCY 11")**

**stop(paste0("\n\n================\n\n", tempo.cat, "\n\n================\n\n", ifelse(is.null(warn), "", paste0("IN ADDITION\nWARNING", ifelse(warn.count > 1, "S", ""), ":\n\n", warn))), call. = FALSE)** *# == in stop() to be able to add several messages between ==*

**}**

**if(any(data1.non.signif.dot$DOT\_NB %in% data1.incon.dot$DOT\_NB)){**

**tempo.cat <- paste0("ERROR IN ", function.name, ": CODE INCONSISTENCY 12")**

**stop(paste0("\n\n================\n\n", tempo.cat, "\n\n================\n\n", ifelse(is.null(warn), "", paste0("IN ADDITION\nWARNING", ifelse(warn.count > 1, "S", ""), ":\n\n", warn))), call. = FALSE)** *# == in stop() to be able to add several messages between ==*

**}**

**if(any(data1.incon.dot$DOT\_NB %in% data1.non.signif.dot$DOT\_NB)){**

**tempo.cat <- paste0("ERROR IN ", function.name, ": CODE INCONSISTENCY 13")**

**stop(paste0("\n\n================\n\n", tempo.cat, "\n\n================\n\n", ifelse(is.null(warn), "", paste0("IN ADDITION\nWARNING", ifelse(warn.count > 1, "S", ""), ":\n\n", warn))), call. = FALSE)** *# == in stop() to be able to add several messages between ==*

**}**

**if( ! is.null(data2)){**

**if(any(data2.signif.dot$DOT\_NB %in% data2.non.signif.dot$DOT\_NB)){**

**tempo.cat <- paste0("ERROR IN ", function.name, ": CODE INCONSISTENCY 14")**

**stop(paste0("\n\n================\n\n", tempo.cat, "\n\n================\n\n", ifelse(is.null(warn), "", paste0("IN ADDITION\nWARNING", ifelse(warn.count > 1, "S", ""), ":\n\n", warn))), call. = FALSE)** *# == in stop() to be able to add several messages between ==*

**}**

**if(any(data2.non.signif.dot$DOT\_NB %in% data2.signif.dot$DOT\_NB)){**

**tempo.cat <- paste0("ERROR IN ", function.name, ": CODE INCONSISTENCY 15")**

**stop(paste0("\n\n================\n\n", tempo.cat, "\n\n================\n\n", ifelse(is.null(warn), "", paste0("IN ADDITION\nWARNING", ifelse(warn.count > 1, "S", ""), ":\n\n", warn))), call. = FALSE)** *# == in stop() to be able to add several messages between ==*

**}**

**if(any(data2.signif.dot$DOT\_NB %in% data2.unknown.dot$DOT\_NB)){**

**tempo.cat <- paste0("ERROR IN ", function.name, ": CODE INCONSISTENCY 16")**

**stop(paste0("\n\n================\n\n", tempo.cat, "\n\n================\n\n", ifelse(is.null(warn), "", paste0("IN ADDITION\nWARNING", ifelse(warn.count > 1, "S", ""), ":\n\n", warn))), call. = FALSE)** *# == in stop() to be able to add several messages between ==*

**}**

**if(any(data2.unknown.dot$DOT\_NB %in% data2.signif.dot$DOT\_NB)){**

**tempo.cat <- paste0("ERROR IN ", function.name, ": CODE INCONSISTENCY 17")**

**stop(paste0("\n\n================\n\n", tempo.cat, "\n\n================\n\n", ifelse(is.null(warn), "", paste0("IN ADDITION\nWARNING", ifelse(warn.count > 1, "S", ""), ":\n\n", warn))), call. = FALSE)** *# == in stop() to be able to add several messages between ==*

**}**

**if(any(data2.signif.dot$DOT\_NB %in% data2.incon.dot$DOT\_NB)){**

**tempo.cat <- paste0("ERROR IN ", function.name, ": CODE INCONSISTENCY 18")**

**stop(paste0("\n\n================\n\n", tempo.cat, "\n\n================\n\n", ifelse(is.null(warn), "", paste0("IN ADDITION\nWARNING", ifelse(warn.count > 1, "S", ""), ":\n\n", warn))), call. = FALSE)** *# == in stop() to be able to add several messages between ==*

**}**

**if(any(data2.incon.dot$DOT\_NB %in% data2.signif.dot$DOT\_NB)){**

**tempo.cat <- paste0("ERROR IN ", function.name, ": CODE INCONSISTENCY 19")**

**stop(paste0("\n\n================\n\n", tempo.cat, "\n\n================\n\n", ifelse(is.null(warn), "", paste0("IN ADDITION\nWARNING", ifelse(warn.count > 1, "S", ""), ":\n\n", warn))), call. = FALSE)** *# == in stop() to be able to add several messages between ==*

**}**

**if(any(data2.non.signif.dot$DOT\_NB %in% data2.unknown.dot$DOT\_NB)){**

**tempo.cat <- paste0("ERROR IN ", function.name, ": CODE INCONSISTENCY 20")**

**stop(paste0("\n\n================\n\n", tempo.cat, "\n\n================\n\n", ifelse(is.null(warn), "", paste0("IN ADDITION\nWARNING", ifelse(warn.count > 1, "S", ""), ":\n\n", warn))), call. = FALSE)** *# == in stop() to be able to add several messages between ==*

**}**

**if(any(data2.unknown.dot$DOT\_NB %in% data2.non.signif.dot$DOT\_NB)){**

**tempo.cat <- paste0("ERROR IN ", function.name, ": CODE INCONSISTENCY 21")**

**stop(paste0("\n\n================\n\n", tempo.cat, "\n\n================\n\n", ifelse(is.null(warn), "", paste0("IN ADDITION\nWARNING", ifelse(warn.count > 1, "S", ""), ":\n\n", warn))), call. = FALSE)** *# == in stop() to be able to add several messages between ==*

**}**

**if(any(data2.non.signif.dot$DOT\_NB %in% data2.incon.dot$DOT\_NB)){**

**tempo.cat <- paste0("ERROR IN ", function.name, ": CODE INCONSISTENCY 22")**

**stop(paste0("\n\n================\n\n", tempo.cat, "\n\n================\n\n", ifelse(is.null(warn), "", paste0("IN ADDITION\nWARNING", ifelse(warn.count > 1, "S", ""), ":\n\n", warn))), call. = FALSE)** *# == in stop() to be able to add several messages between ==*

**}**

**if(any(data2.incon.dot$DOT\_NB %in% data2.non.signif.dot$DOT\_NB)){**

**tempo.cat <- paste0("ERROR IN ", function.name, ": CODE INCONSISTENCY 23")**

**stop(paste0("\n\n================\n\n", tempo.cat, "\n\n================\n\n", ifelse(is.null(warn), "", paste0("IN ADDITION\nWARNING", ifelse(warn.count > 1, "S", ""), ":\n\n", warn))), call. = FALSE)** *# == in stop() to be able to add several messages between ==*

**}**

**if(any(data2.unknown.dot$DOT\_NB %in% data2.incon.dot$DOT\_NB)){**

**tempo.cat <- paste0("ERROR IN ", function.name, ": CODE INCONSISTENCY 24")**

**stop(paste0("\n\n================\n\n", tempo.cat, "\n\n================\n\n", ifelse(is.null(warn), "", paste0("IN ADDITION\nWARNING", ifelse(warn.count > 1, "S", ""), ":\n\n", warn))), call. = FALSE)** *# == in stop() to be able to add several messages between ==*

**}**

**if(any(data2.incon.dot$DOT\_NB %in% data2.unknown.dot$DOT\_NB)){**

**tempo.cat <- paste0("ERROR IN ", function.name, ": CODE INCONSISTENCY 25")**

**stop(paste0("\n\n================\n\n", tempo.cat, "\n\n================\n\n", ifelse(is.null(warn), "", paste0("IN ADDITION\nWARNING", ifelse(warn.count > 1, "S", ""), ":\n\n", warn))), call. = FALSE)** *# == in stop() to be able to add several messages between ==*

**}**

**}**

*# end verif*

*# plot*

*# recovering the axes data whatever plot or not*

**if(is.null(data2)){**

**axes <- fun\_gg\_scatter(data1 = list(data1), x = list(x1), y = list(y1), categ = list(NULL), color = list(fun\_gg\_palette(2)[2]), geom = list("geom\_point"), alpha = list(0.5), x.lim = x.range.plot, y.lim = y.range.plot, raster = raster, plot = FALSE, return = TRUE)$axes**

**}else{**

**axes <- fun\_gg\_scatter(data1 = list(data1, data2), x = list(x1, x2), y = list(y1, y2), categ = list(NULL, NULL), color = list(fun\_gg\_palette(2)[2], fun\_gg\_palette(2)[1]), geom = list("geom\_point", "geom\_point"), alpha = list(0.5, 0.5), x.lim = x.range.plot, y.lim = y.range.plot, raster = raster, plot = FALSE, return = TRUE)$axes**

**}**

*# end recovering the axes data whatever plot or not*

**if(plot == TRUE){**

*# add a categ for plot legend*

**tempo.df.name <- c("data1", "data1.signif.dot", "data1.incon.dot", "data2", "data2.signif.dot", "data2.unknown.dot", "data2.incon.dot")**

**tempo.class.name <- c("data1", "data1", "data1", "data2", "data2", "data2", "data2")**

**for(i2 in 1:length(tempo.df.name)){**

**if( ! is.null(get(tempo.df.name[i2]))){**

**assign(tempo.df.name[i2], data.frame(get(tempo.df.name[i2]), kind = tempo.class.name[i2]))**

**}**

**}**

*# end add a categ for plot legend*

**if(( ! is.null(x.range.split)) & ( ! is.null(y.range.split))){**

**if(graph.in.file == FALSE){**

**fun\_open(pdf = FALSE)**

**}**

**tempo.graph <- fun\_gg\_scatter(data1 = list(data1, hframe, vframe), x = list(x1, "x", "x"), y = list(y1, "y", "y"), categ = list("kind", "kind", "kind"), legend.name = list("DATASET", "HORIZ FRAME" , "VERT FRAME"), 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"), alpha = list(0.5, 0.5, 0.5), title = "DATA1", x.lim = x.range.plot, y.lim = y.range.plot, raster = raster, return = TRUE)**

**if( ! is.null(tempo.graph$warn)){**

**warn <- paste0(ifelse(is.null(warn), tempo.graph$warn, paste0(warn, "\n", tempo.graph$warn)))**

**}**

**if( ! is.null(data1.signif.dot)){**

**if(graph.in.file == FALSE){**

**fun\_open(pdf = FALSE)**

**}**

**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("kind", "kind", "kind", "kind"), legend.name = list("DATASET", "HORIZ FRAME" , "VERT FRAME", "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"), alpha = list(0.5, 0.5, 0.5, 0.5), title = "DATA1 + DATA1 SIGNIFICANT DOTS", x.lim = x.range.plot, y.lim = y.range.plot, raster = raster, return = TRUE)**

**if( ! is.null(tempo.graph$warn)){**

**warn <- paste0(ifelse(is.null(warn), tempo.graph$warn, paste0(warn, "\n", tempo.graph$warn)))**

**}**

**}else{**

**if(graph.in.file == FALSE){**

**fun\_open(pdf = FALSE)**

**}**

**fun\_gg\_empty\_graph(text = "NO PLOT\nBECAUSE\nNO DATA1 DOTS\nOUTSIDE THE FRAMES", text.size = 8, title = "DATA1 + DATA1 SIGNIFICANT DOTS")**

**}**

**if( ! is.null(data1.incon.dot)){**

**if(graph.in.file == FALSE){**

**fun\_open(pdf = FALSE)**

**}**

**tempo.graph <- fun\_gg\_scatter(data1 = list(data1, hframe, vframe, data1.incon.dot), x = list(x1, "x", "x", x1), y = list(y1, "y", "y", y1), categ = list("kind", "kind", "kind", "kind"), legend.name = list("DATASET", "HORIZ FRAME" , "VERT FRAME", "INCONSISTENT 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), fun\_gg\_palette(7)[6]), geom = list("geom\_point", "geom\_path", "geom\_path", "geom\_point"), alpha = list(0.5, 0.5, 0.5, 0.5), title = "DATA1 + DATA1 INCONSISTENT DOTS", x.lim = x.range.plot, y.lim = y.range.plot, raster = raster, return = TRUE)**

**if( ! is.null(tempo.graph$warn)){**

**warn <- paste0(ifelse(is.null(warn), tempo.graph$warn, paste0(warn, "\n", tempo.graph$warn)))**

**}**

**}else{**

**if(graph.in.file == FALSE){**

**fun\_open(pdf = FALSE)**

**}**

**fun\_gg\_empty\_graph(text = "NO PLOT\nBECAUSE\nNO DATA1\nINCONSISTENT DOTS", text.size = 8, title = "DATA1 + DATA1 INCONSISTENT DOTS")**

**}**

**if( ! is.null(data2)){**

**if(graph.in.file == FALSE){**

**fun\_open(pdf = FALSE)**

**}**

**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("kind", "kind", "kind", "kind"), legend.name = list("DATASET", "DATASET", "HORIZ FRAME" , "VERT FRAME"), 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"), alpha = list(0.5, 0.5, 0.5, 0.5), title = "DATA1 + DATA2", x.lim = x.range.plot, y.lim = y.range.plot, raster = raster, return = TRUE)**

**if( ! is.null(tempo.graph$warn)){**

**warn <- paste0(ifelse(is.null(warn), tempo.graph$warn, paste0(warn, "\n", tempo.graph$warn)))**

**}**

**if( ! is.null(data2.signif.dot)){**

**if(graph.in.file == FALSE){**

**fun\_open(pdf = FALSE)**

**}**

**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("kind", "kind", "kind", "kind", "kind"), legend.name = list("DATASET", "DATASET", "SIGNIF DOTS", "HORIZ FRAME" , "VERT FRAME"), 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"), alpha = list(0.5, 0.5, 0.5, 0.5, 0.5), title = "DATA1 + DATA2 + DATA2 SIGNIFICANT DOTS", x.lim = x.range.plot, y.lim = y.range.plot, raster = raster, return = TRUE)**

**if( ! is.null(tempo.graph$warn)){**

**warn <- paste0(ifelse(is.null(warn), tempo.graph$warn, paste0(warn, "\n", tempo.graph$warn)))**

**}**

**}else{**

**if(graph.in.file == FALSE){**

**fun\_open(pdf = FALSE)**

**}**

**fun\_gg\_empty\_graph(text = "NO PLOT\nBECAUSE\nNO DATA2 DOTS\nOUTSIDE THE FRAMES", text.size = 8, title = "DATA1 + DATA2 + DATA2 SIGNIFICANT DOTS")**

**}**

**if( ! is.null(data2.incon.dot)){**

**if(graph.in.file == FALSE){**

**fun\_open(pdf = FALSE)**

**}**

**tempo.graph <- fun\_gg\_scatter(data1 = list(data1, data2, data2.incon.dot, hframe , vframe), x = list(x1, x2, x2, "x", "x"), y = list(y1, y2, y2, "y", "y"), categ = list("kind", "kind", "kind", "kind", "kind"), legend.name = list("DATASET", "DATASET", "INCONSISTENT DOTS", "HORIZ FRAME" , "VERT FRAME"), color = list(fun\_gg\_palette(2)[2], fun\_gg\_palette(2)[1], fun\_gg\_palette(7)[6], 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"), alpha = list(0.5, 0.5, 0.5, 0.5, 0.5), title = "DATA1 + DATA2 + DATA2 INCONSISTENT DOTS", x.lim = x.range.plot, y.lim = y.range.plot, raster = raster, return = TRUE)**

**if( ! is.null(tempo.graph$warn)){**

**warn <- paste0(ifelse(is.null(warn), tempo.graph$warn, paste0(warn, "\n", tempo.graph$warn)))**

**}**

**}else{**

**if(graph.in.file == FALSE){**

**fun\_open(pdf = FALSE)**

**}**

**fun\_gg\_empty\_graph(text = "NO PLOT\nBECAUSE\nNO DATA2\nINCONSISTENT DOTS", text.size = 8, title = "DATA2 + DATA2 INCONSISTENT DOTS")**

**}**

**if( ! is.null(data2.unknown.dot)){**

**if(graph.in.file == FALSE){**

**fun\_open(pdf = FALSE)**

**}**

**tempo.graph <- fun\_gg\_scatter(data1 = list(data1, data2, data2.unknown.dot, hframe , vframe), x = list(x1, x2, x2, "x", "x"), y = list(y1, y2, y2, "y", "y"), categ = list("kind", "kind", "kind", "kind", "kind"), legend.name = list("DATASET", "DATASET", "UNKNOWN DOTS", "HORIZ FRAME" , "VERT FRAME"), color = list(fun\_gg\_palette(2)[2], fun\_gg\_palette(2)[1], fun\_gg\_palette(7)[5], 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"), alpha = list(0.5, 0.5, 0.5, 0.5, 0.5), title = "DATA1 + DATA2 + DATA2 UNKNOWN DOTS", x.lim = x.range.plot, y.lim = y.range.plot, raster = raster, return = TRUE)**

**if( ! is.null(tempo.graph$warn)){**

**warn <- paste0(ifelse(is.null(warn), tempo.graph$warn, paste0(warn, "\n", tempo.graph$warn)))**

**}**

**}else{**

**if(graph.in.file == FALSE){**

**fun\_open(pdf = FALSE)**

**}**

**fun\_gg\_empty\_graph(text = "NO PLOT\nBECAUSE\nNO DATA2\nUNKNOWN DOTS", text.size = 12, title = "DATA2 + DATA2 UNKNOWN DOTS")**

**}**

**}**

**}else if(( ! is.null(x.range.split)) & is.null(y.range.split)){**

**if(graph.in.file == FALSE){**

**fun\_open(pdf = FALSE)**

**}**

**tempo.graph <- fun\_gg\_scatter(data1 = list(data1, hframe), x = list(x1, "x"), y = list(y1, "y"), categ = list("kind", "kind"), legend.name = list("DATASET", "HORIZ FRAME"), 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"), alpha = list(0.5, 0.5), title = "DATA1", x.lim = x.range.plot, y.lim = y.range.plot, raster = raster, return = TRUE)**

**if( ! is.null(tempo.graph$warn)){**

**warn <- paste0(ifelse(is.null(warn), tempo.graph$warn, paste0(warn, "\n", tempo.graph$warn)))**

**}**

**if( ! is.null(data1.signif.dot)){**

**if(graph.in.file == FALSE){**

**fun\_open(pdf = FALSE)**

**}**

**tempo.graph <- fun\_gg\_scatter(data1 = list(data1, hframe, data1.signif.dot), x = list(x1, "x", x1), y = list(y1, "y", y1), categ = list("kind", "kind", "kind"), legend.name = list("DATASET", "HORIZ FRAME", "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"), alpha = list(0.5, 0.5, 0.5), title = "DATA1 + DATA1 SIGNIFICANT DOTS", x.lim = x.range.plot, y.lim = y.range.plot, raster = raster, return = TRUE)**

**if( ! is.null(tempo.graph$warn)){**

**warn <- paste0(ifelse(is.null(warn), tempo.graph$warn, paste0(warn, "\n", tempo.graph$warn)))**

**}**

**}else{**

**if(graph.in.file == FALSE){**

**fun\_open(pdf = FALSE)**

**}**

**fun\_gg\_empty\_graph(text = "NO PLOT\nBECAUSE\nNO DATA1 DOTS\nOUTSIDE THE FRAMES", text.size = 8, title = "DATA1 + DATA1 SIGNIFICANT DOTS")**

**}**

**if( ! is.null(data1.incon.dot)){**

**if(graph.in.file == FALSE){**

**fun\_open(pdf = FALSE)**

**}**

**tempo.graph <- fun\_gg\_scatter(data1 = list(data1, hframe, data1.incon.dot), x = list(x1, "x", x1), y = list(y1, "y", y1), categ = list("kind", "kind", "kind"), legend.name = list("DATASET", "HORIZ FRAME", "INCONSISTENT DOTS"), color = list(fun\_gg\_palette(2)[2], rep(hsv(h = c(0.1, 0.15), v = c(0.75, 1)), 2), fun\_gg\_palette(7)[6]), geom = list("geom\_point", "geom\_path", "geom\_point"), alpha = list(0.5, 0.5, 0.5), title = "DATA1 + DATA1 INCONSISTENT DOTS", x.lim = x.range.plot, y.lim = y.range.plot, raster = raster, return = TRUE)**

**if( ! is.null(tempo.graph$warn)){**

**warn <- paste0(ifelse(is.null(warn), tempo.graph$warn, paste0(warn, "\n", tempo.graph$warn)))**

**}**

**}else{**

**if(graph.in.file == FALSE){**

**fun\_open(pdf = FALSE)**

**}**

**fun\_gg\_empty\_graph(text = "NO PLOT\nBECAUSE\nNO DATA1\nINCONSISTENT DOTS", text.size = 8, title = "DATA1 + DATA1 INCONSISTENT DOTS")**

**}**

**if( ! is.null(data2)){**

**if(graph.in.file == FALSE){**

**fun\_open(pdf = FALSE)**

**}**

**tempo.graph <- fun\_gg\_scatter(data1 = list(data1, data2, hframe), x = list(x1, x2, "x"), y = list(y1, y2, "y"), categ = list("kind", "kind", "kind"), legend.name = list("DATASET", "DATASET", "HORIZ FRAME"), 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"), alpha = list(0.5, 0.5, 0.5), title = "DATA1 + DATA2", x.lim = x.range.plot, y.lim = y.range.plot, raster = raster, return = TRUE)**

**if( ! is.null(tempo.graph$warn)){**

**warn <- paste0(ifelse(is.null(warn), tempo.graph$warn, paste0(warn, "\n", tempo.graph$warn)))**

**}**

**if( ! is.null(data2.signif.dot)){**

**if(graph.in.file == FALSE){**

**fun\_open(pdf = FALSE)**

**}**

**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("kind", "kind", "kind", "kind"), legend.name = list("DATASET", "DATASET", "SIGNIF DOTS", "HORIZ FRAME"), 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"), alpha = list(0.5, 0.5, 0.5, 0.5), title = "DATA1 + DATA2 + DATA2 SIGNIFICANT DOTS", x.lim = x.range.plot, y.lim = y.range.plot, raster = raster, return = TRUE)**

**if( ! is.null(tempo.graph$warn)){**

**warn <- paste0(ifelse(is.null(warn), tempo.graph$warn, paste0(warn, "\n", tempo.graph$warn)))**

**}**

**}else{**

**if(graph.in.file == FALSE){**

**fun\_open(pdf = FALSE)**

**}**

**fun\_gg\_empty\_graph(text = "NO PLOT\nBECAUSE\nNO DATA2 DOTS\nOUTSIDE THE FRAMES", text.size = 8, title = "DATA1 + DATA2 + DATA2 SIGNIFICANT DOTS")**

**}**

**if( ! is.null(data2.incon.dot)){**

**if(graph.in.file == FALSE){**

**fun\_open(pdf = FALSE)**

**}**

**tempo.graph <- fun\_gg\_scatter(data1 = list(data1, data2, data2.incon.dot, hframe), x = list(x1, x2, x2, "x"), y = list(y1, y2, y2, "y"), categ = list("kind", "kind", "kind", "kind"), legend.name = list("DATASET", "DATASET", "INCONSISTENT DOTS", "HORIZ FRAME"), color = list(fun\_gg\_palette(2)[2], fun\_gg\_palette(2)[1], fun\_gg\_palette(7)[6], rep(hsv(h = c(0.1, 0.15), v = c(0.75, 1)), 2)), geom = list("geom\_point", "geom\_point", "geom\_point", "geom\_path"), alpha = list(0.5, 0.5, 0.5, 0.5), title = "DATA1 + DATA2 + DATA2 INCONSISTENT DOTS", x.lim = x.range.plot, y.lim = y.range.plot, raster = raster, return = TRUE)**

**if( ! is.null(tempo.graph$warn)){**

**warn <- paste0(ifelse(is.null(warn), tempo.graph$warn, paste0(warn, "\n", tempo.graph$warn)))**

**}**

**}else{**

**if(graph.in.file == FALSE){**

**fun\_open(pdf = FALSE)**

**}**

**fun\_gg\_empty\_graph(text = "NO PLOT\nBECAUSE\nNO DATA2\nINCONSISTENT DOTS", text.size = 8, title = "DATA2 + DATA2 INCONSISTENT DOTS")**

**}**

**if( ! is.null(data2.unknown.dot)){**

**if(graph.in.file == FALSE){**

**fun\_open(pdf = FALSE)**

**}**

**tempo.graph <- fun\_gg\_scatter(data1 = list(data1, data2, data2.unknown.dot, hframe), x = list(x1, x2, x2, "x"), y = list(y1, y2, y2, "y"), categ = list("kind", "kind", "kind", "kind"), legend.name = list("DATASET", "DATASET", "UNKNOWN DOTS", "HORIZ FRAME"), color = list(fun\_gg\_palette(2)[2], fun\_gg\_palette(2)[1], fun\_gg\_palette(7)[5], rep(hsv(h = c(0.1, 0.15), v = c(0.75, 1)), 2)), geom = list("geom\_point", "geom\_point", "geom\_point", "geom\_path"), alpha = list(0.5, 0.5, 0.5, 0.5), title = "DATA1 + DATA2 + DATA2 UNKNOWN DOTS", x.lim = x.range.plot, y.lim = y.range.plot, raster = raster, return = TRUE)**

**if( ! is.null(tempo.graph$warn)){**

**warn <- paste0(ifelse(is.null(warn), tempo.graph$warn, paste0(warn, "\n", tempo.graph$warn)))**

**}**

**}else{**

**if(graph.in.file == FALSE){**

**fun\_open(pdf = FALSE)**

**}**

**fun\_gg\_empty\_graph(text = "NO PLOT\nBECAUSE\nNO DATA2\nUNKNOWN DOTS", text.size = 8, title = "DATA2 + DATA2 UNKNOWN DOTS")**

**}**

**}**

**}else if(is.null(x.range.split) & ( ! is.null(y.range.split))){**

**if(graph.in.file == FALSE){**

**fun\_open(pdf = FALSE)**

**}**

**tempo.graph <- fun\_gg\_scatter(data1 = list(data1, vframe), x = list(x1, "x"), y = list(y1, "y"), categ = list("kind", "kind"), legend.name = list("DATASET", "VERT FRAME"), 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"), alpha = list(0.5, 0.5), title = "DATA1", x.lim = x.range.plot, y.lim = y.range.plot, raster = raster, return = TRUE)**

**if( ! is.null(tempo.graph$warn)){**

**warn <- paste0(ifelse(is.null(warn), tempo.graph$warn, paste0(warn, "\n", tempo.graph$warn)))**

**}**

**if( ! is.null(data1.signif.dot)){**

**if(graph.in.file == FALSE){**

**fun\_open(pdf = FALSE)**

**}**

**tempo.graph <- fun\_gg\_scatter(data1 = list(data1, vframe, data1.signif.dot), x = list(x1, "x", x1), y = list(y1, "y", y1), categ = list("kind", "kind", "kind"), legend.name = list("DATASET", "VERT FRAME", "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"), alpha = list(0.5, 0.5, 0.5), title = "DATA1 + DATA1 SIGNIFICANT DOTS", x.lim = x.range.plot, y.lim = y.range.plot, raster = raster, return = TRUE)**

**if( ! is.null(tempo.graph$warn)){**

**warn <- paste0(ifelse(is.null(warn), tempo.graph$warn, paste0(warn, "\n", tempo.graph$warn)))**

**}**

**}else{**

**if(graph.in.file == FALSE){**

**fun\_open(pdf = FALSE)**

**}**

**fun\_gg\_empty\_graph(text = "NO PLOT\nBECAUSE\nNO DATA1 DOTS\nOUTSIDE THE FRAMES", text.size = 8, title = "DATA1 + DATA1 SIGNIFICANT DOTS")**

**}**

**if( ! is.null(data1.incon.dot)){**

**if(graph.in.file == FALSE){**

**fun\_open(pdf = FALSE)**

**}**

**tempo.graph <- fun\_gg\_scatter(data1 = list(data1, vframe, data1.incon.dot), x = list(x1, "x", x1), y = list(y1, "y", y1), categ = list("kind", "kind", "kind"), legend.name = list("DATASET", "VERT FRAME", "INCONSISTENT DOTS"), color = list(fun\_gg\_palette(2)[2], rep(hsv(h = c(0.5, 0.6), v = c(0.9, 1)), 2), fun\_gg\_palette(7)[6]), geom = list("geom\_point", "geom\_path", "geom\_point"), alpha = list(0.5, 0.5, 0.5), title = "DATA1 + DATA1 INCONSISTENT DOTS", x.lim = x.range.plot, y.lim = y.range.plot, raster = raster, return = TRUE)**

**if( ! is.null(tempo.graph$warn)){**

**warn <- paste0(ifelse(is.null(warn), tempo.graph$warn, paste0(warn, "\n", tempo.graph$warn)))**

**}**

**}else{**

**if(graph.in.file == FALSE){**

**fun\_open(pdf = FALSE)**

**}**

**fun\_gg\_empty\_graph(text = "NO PLOT\nBECAUSE\nNO DATA1\nINCONSISTENT DOTS", text.size = 8, title = "DATA1 + DATA1 INCONSISTENT DOTS")**

**}**

**if( ! is.null(data2)){**

**if(graph.in.file == FALSE){**

**fun\_open(pdf = FALSE)**

**}**

**tempo.graph <- fun\_gg\_scatter(data1 = list(data1, data2, vframe), x = list(x1, x2, "x"), y = list(y1, y2, "y"), categ = list("kind", "kind", "kind"), legend.name = list("DATASET", "DATASET", "VERT FRAME"), 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"), alpha = list(0.5, 0.5, 0.5), title = "DATA1 + DATA2", x.lim = x.range.plot, y.lim = y.range.plot, raster = raster, return = TRUE)**

**if( ! is.null(tempo.graph$warn)){**

**warn <- paste0(ifelse(is.null(warn), tempo.graph$warn, paste0(warn, "\n", tempo.graph$warn)))**

**}**

**if( ! is.null(data2.signif.dot)){**

**if(graph.in.file == FALSE){**

**fun\_open(pdf = FALSE)**

**}**

**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("kind", "kind", "kind", "kind"), legend.name = list("DATASET", "DATASET", "SIGNIF DOTS", "VERT FRAME"), 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"), alpha = list(0.5, 0.5, 0.5, 0.5), title = "DATA1 + DATA2 + DATA2 SIGNIFICANT DOTS", x.lim = x.range.plot, y.lim = y.range.plot, raster = raster, return = TRUE)**

**if( ! is.null(tempo.graph$warn)){**

**warn <- paste0(ifelse(is.null(warn), tempo.graph$warn, paste0(warn, "\n", tempo.graph$warn)))**

**}**

**}else{**

**if(graph.in.file == FALSE){**

**fun\_open(pdf = FALSE)**

**}**

**fun\_gg\_empty\_graph(text = "NO PLOT\nBECAUSE\nNO DATA2 DOTS\nOUTSIDE THE FRAMES", text.size = 8, title = "DATA1 + DATA2 + DATA2 SIGNIFICANT DOTS")**

**}**

**if( ! is.null(data2.incon.dot)){**

**if(graph.in.file == FALSE){**

**fun\_open(pdf = FALSE)**

**}**

**tempo.graph <- fun\_gg\_scatter(data1 = list(data1, data2, data2.incon.dot, vframe), x = list(x1, x2, x2, "x"), y = list(y1, y2, y2, "y"), categ = list("kind", "kind", "kind", "kind"), legend.name = list("DATASET", "DATASET", "INCONSISTENT DOTS", "VERT FRAME"), color = list(fun\_gg\_palette(2)[2], fun\_gg\_palette(2)[1], fun\_gg\_palette(7)[6], rep(hsv(h = c(0.5, 0.6), v = c(0.9, 1)), 2)), geom = list("geom\_point", "geom\_point", "geom\_point", "geom\_path"), alpha = list(0.5, 0.5, 0.5, 0.5), title = "DATA1 + DATA2 + DATA2 INCONSISTENT DOTS", x.lim = x.range.plot, y.lim = y.range.plot, raster = raster, return = TRUE)**

**if( ! is.null(tempo.graph$warn)){**

**warn <- paste0(ifelse(is.null(warn), tempo.graph$warn, paste0(warn, "\n", tempo.graph$warn)))**

**}**

**}else{**

**if(graph.in.file == FALSE){**

**fun\_open(pdf = FALSE)**

**}**

**fun\_gg\_empty\_graph(text = "NO PLOT\nBECAUSE\nNO DATA2\nINCONSISTENT DOTS", text.size = 8, title = "DATA2 + DATA2 INCONSISTENT DOTS")**

**}**

**if( ! is.null(data2.unknown.dot)){**

**if(graph.in.file == FALSE){**

**fun\_open(pdf = FALSE)**

**}**

**tempo.graph <- fun\_gg\_scatter(data1 = list(data1, data2, data2.unknown.dot, vframe), x = list(x1, x2, x2, "x"), y = list(y1, y2, y2, "y"), categ = list("kind", "kind", "kind", "kind"), legend.name = list("DATASET", "DATASET", "UNKNOWN DOTS", "VERT FRAME"), color = list(fun\_gg\_palette(2)[2], fun\_gg\_palette(2)[1], fun\_gg\_palette(7)[5], rep(hsv(h = c(0.5, 0.6), v = c(0.9, 1)), 2)), geom = list("geom\_point", "geom\_point", "geom\_point", "geom\_path"), alpha = list(0.5, 0.5, 0.5, 0.5), title = "DATA1 + DATA2 + DATA2 UNKNOWN DOTS", x.lim = x.range.plot, y.lim = y.range.plot, raster = raster, return = TRUE)**

**if( ! is.null(tempo.graph$warn)){**

**warn <- paste0(ifelse(is.null(warn), tempo.graph$warn, paste0(warn, "\n", tempo.graph$warn)))**

**}**

**}else{**

**if(graph.in.file == FALSE){**

**fun\_open(pdf = FALSE)**

**}**

**fun\_gg\_empty\_graph(text = "NO PLOT\nBECAUSE\nNO DATA2\nUNKNOWN DOTS", text.size = 8, title = "DATA2 + DATA2 UNKNOWN DOTS")**

**}**

**}**

**}**

**}**

*# end plot*

**if(warn.print == TRUE & ! is.null(warn)){**

**warning(paste0("FROM ", function.name, ":\n\n", warn), call. = FALSE)**

**}**

**tempo.list <- list(data1.removed.row.nb = data1.removed.row.nb, data1.removed.rows = data1.removed.rows, data2.removed.row.nb = data2.removed.row.nb, data2.removed.rows = data2.removed.rows, hframe = hframe, vframe = vframe, data1.signif.dot = data1.signif.dot, data1.non.signif.dot = data1.non.signif.dot, data1.inconsistent.dot = data1.incon.dot, data2.signif.dot = data2.signif.dot, data2.non.signif.dot = data2.non.signif.dot, data2.unknown.dot = data2.unknown.dot, data2.inconsistent.dot = data2.incon.dot, axes = axes, warn = warn)**

**return(tempo.list)**

**}**

################ Import

######## fun\_pack() #### check if R packages are present and import into the working environment

**# Check OK: clear to go Apollo**

**fun\_pack <- function(req.package, load = FALSE, lib.path = 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*

*# load: logical. Load the package into the environement (using library())? Interesting if packages are not in default folders or for checking the functions names of packages using search()*

*# lib.path: optional character vector specifying the absolute pathways of the directories containing some of the listed packages in the req.package argument, if not in the default directories. Ignored if NULL*

*# REQUIRED PACKAGES*

*# none*

*# REQUIRED FUNCTIONS FROM CUTE\_LITTLE\_R\_FUNCTION*

*# fun\_check()*

*# RETURN*

*# nothing*

*# EXAMPLES*

*# fun\_pack(req.package = "nopackage")*

*# fun\_pack(req.package = "ggplot2")*

*# fun\_pack(req.package = "ggplot2", lib.path = "blablabla")*

*# DEBUGGING*

*# req.package = "ggplot2" ; lib.path = "C:/Program Files/R/R-3.5.1/library"*

*# req.package = "serpentine" ; lib.path = "C:/users/gael/appdata/roaming/python/python36/site-packages"*

*# function name*

**function.name <- paste0(as.list(match.call(expand.dots=FALSE))[[1]], "()")**

*# end function name*

*# required function checking*

**if(length(utils::find("fun\_check", mode = "function")) == 0){**

**tempo.cat <- paste0("ERROR IN ", function.name, ": REQUIRED fun\_check() FUNCTION IS MISSING IN THE R ENVIRONMENT")**

**stop(paste0("\n\n================\n\n", tempo.cat, "\n\n================\n\n"), call. = FALSE)** *# == in stop() to be able to add several messages between ==*

**}**

*# end required function checking*

*# argument checking*

**arg.check <- NULL** *#*

**text.check <- NULL** *#*

**checked.arg.names <- NULL** *# for function debbuging: used by r\_debugging\_tools*

**ee <- expression(arg.check <- c(arg.check, tempo$problem) , text.check <- c(text.check, tempo$text) , checked.arg.names <- c(checked.arg.names, tempo$fun.name))**

**tempo <- fun\_check(data = req.package, class = "vector", mode = "character", fun.name = function.name) ; eval(ee)**

**tempo <- fun\_check(data = load, class = "vector", mode = "logical", length = 1, fun.name = function.name) ; eval(ee)**

**if( ! is.null(lib.path)){**

**tempo <- fun\_check(data = lib.path, class = "vector", mode = "character", fun.name = function.name) ; eval(ee)**

**if(tempo$problem == FALSE){**

**if( ! all(dir.exists(lib.path))){** *# separation to avoid the problem of tempo$problem == FALSE and lib.path == NA*

**tempo.cat <- paste0("ERROR IN ", function.name, ": DIRECTORY PATH INDICATED IN THE lib.path ARGUMENT DOES NOT EXISTS:\n", paste(lib.path, collapse = "\n"))**

**text.check <- c(text.check, tempo.cat)**

**arg.check <- c(arg.check, TRUE)**

**}**

**}**

**}**

**if(any(arg.check) == TRUE){**

**stop(paste0("\n\n================\n\n", paste(text.check[arg.check], collapse = "\n"), "\n\n================\n\n"), call. = FALSE)** *#*

**}**

*# source("C:/Users/Gael/Documents/Git\_versions\_to\_use/debugging\_tools\_for\_r\_dev-v1.2/r\_debugging\_tools-v1.2.R") ; eval(parse(text = str\_basic\_arg\_check\_dev)) ; eval(parse(text = str\_arg\_check\_with\_fun\_check\_dev)) # activate this line and use the function (with no arguments left as NULL) to check arguments status and if they have been checked using fun\_check()*

*# end argument checking*

*# main code*

**if(is.null(lib.path)){**

**lib.path <- .libPaths()** *# .libPaths(new = lib.path) # or .libPaths(new = c(.libPaths(), lib.path))*

**}else{**

**.libPaths(new = sub(x = lib.path, 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*

**}**

**tempo <- NULL**

**for(i1 in 1:length(req.package)){**

**if( ! req.package[i1] %in% rownames(utils::installed.packages(lib.loc = lib.path))){**

**tempo <- c(tempo, req.package[i1])**

**}**

**}**

**if( ! is.null(tempo)){**

**tempo.cat <- paste0(**

**"ERROR IN ",**

**function.name,**

**": PACKAGE",**

**ifelse(length(tempo) == 1, paste0("\n\n", tempo, "\n\n"), paste0("S\n", paste(tempo, collapse = "\n"), "\n")),**

**"MUST BE INSTALLED IN",**

**ifelse(length(lib.path) == 1, "", " ONE OF THESE FOLDERS"),**

**":\n",**

**paste(lib.path, collapse = "\n")**

**)**

**stop(paste0("\n\n================\n\n", tempo.cat, "\n\n================\n\n"), call. = FALSE)** *# == in stop() to be able to add several messages between ==*

**}else if(load == TRUE){**

**suppressMessages(suppressWarnings(suppressPackageStartupMessages(library(req.package[i1], lib.loc = lib.path, quietly = TRUE, character.only = TRUE))))**

**}**

**}**

######## fun\_python\_pack() #### check if python packages are present

**# Check OK: clear to go Apollo**

**fun\_python\_pack <- function(req.package, python.exec.path = NULL, lib.path = NULL, R.lib.path = NULL){**

*# AIM*

*# check if the specified python packages are present in the computer (no import)*

*# WARNINGS*

*# for python 3.7. Previous versions return an error "Error in sys$stdout$flush() : attempt to apply non-function"*

*# ARGUMENTS*

*# req.package: character vector of package names to import*

*# python.exec.path: optional character vector specifying the absolute pathways of the executable python file to use (associated to the packages to use). If NULL, the reticulate::import\_from\_path() function used in fun\_python\_pack() seeks for an available version of python.exe, and then uses python\_config(python\_version, required\_module, python\_versions). But might not be the correct one for the lib.path parameter specified. Thus, it is recommanded to do not leave NULL, notably when using computing clusters*

*# lib.path: optional character vector specifying the absolute pathways of the directories containing some of the listed packages in the req.package argument, if not in the default directories*

*# R.lib.path: absolute path of the reticulate packages, if not in the default folders*

*# REQUIRED PACKAGES*

*# reticulate*

*# REQUIRED FUNCTIONS FROM CUTE\_LITTLE\_R\_FUNCTION*

*# fun\_check()*

*# fun\_pack()*

*# RETURN*

*# nothing*

*# EXAMPLES*

*# example of error message*

*# fun\_python\_pack(req.package = "nopackage")*

*# example without error message (require the installation of the python serpentine package from https://github.com/koszullab/serpentine*

*# fun\_python\_pack(req.package = "serpentine", python.exec.path = "C:/ProgramData/Anaconda3/python.exe", lib.path = "c:/programdata/anaconda3/lib/site-packages/")*

*# another example of error message*

*# fun\_python\_pack(req.package = "serpentine", lib.path = "blablabla")*

*# DEBUGGING*

*# req.package = "serpentine" ; python.exec.path = "C:/ProgramData/Anaconda3/python.exe" ; lib.path = "c:/programdata/anaconda3/lib/site-packages/" ; R.lib.path = NULL*

*# req.package = "bad" ; lib.path = NULL ; R.lib.path = NULL*

*# function name*

**function.name <- paste0(as.list(match.call(expand.dots=FALSE))[[1]], "()")**

*# end function name*

*# required function checking*

**if(length(utils::find("fun\_check", mode = "function")) == 0){**

**tempo.cat <- paste0("ERROR IN ", function.name, ": REQUIRED fun\_check() FUNCTION IS MISSING IN THE R ENVIRONMENT")**

**stop(paste0("\n\n================\n\n", tempo.cat, "\n\n================\n\n"), call. = FALSE)** *# == in stop() to be able to add several messages between ==*

**}**

**if(length(utils::find("fun\_pack", mode = "function")) == 0){**

**tempo.cat <- paste0("ERROR IN ", function.name, ": REQUIRED fun\_pack() FUNCTION IS MISSING IN THE R ENVIRONMENT")**

**stop(paste0("\n\n================\n\n", tempo.cat, "\n\n================\n\n"), call. = FALSE)** *# == in stop() to be able to add several messages between ==*

**}**

*# end required function checking*

*# argument checking*

**arg.check <- NULL** *#*

**text.check <- NULL** *#*

**checked.arg.names <- NULL** *# for function debbuging: used by r\_debugging\_tools*

**ee <- expression(arg.check <- c(arg.check, tempo$problem) , text.check <- c(text.check, tempo$text) , checked.arg.names <- c(checked.arg.names, tempo$fun.name))**

**tempo <- fun\_check(data = req.package, class = "character", fun.name = function.name) ; eval(ee)**

**if( ! is.null(python.exec.path)){**

**tempo <- fun\_check(data = python.exec.path, class = "character", length = 1, fun.name = function.name) ; eval(ee)**

**if(tempo$problem == FALSE){**

**if( ! all(file.exists(python.exec.path))){** *# separation to avoid the problem of tempo$problem == FALSE and python.exec.path == NA*

**tempo.cat <- paste0("ERROR IN ", function.name, ": FILE PATH INDICATED IN THE python.exec.path ARGUMENT DOES NOT EXISTS:\n", paste(python.exec.path, collapse = "\n"))**

**text.check <- c(text.check, tempo.cat)**

**arg.check <- c(arg.check, TRUE)**

**}**

**}**

**}**

**if( ! is.null(lib.path)){**

**tempo <- fun\_check(data = lib.path, class = "vector", mode = "character", fun.name = function.name) ; eval(ee)**

**if(tempo$problem == FALSE){**

**if( ! all(dir.exists(lib.path))){** *# separation to avoid the problem of tempo$problem == FALSE and lib.path == NA*

**tempo.cat <- paste0("ERROR IN ", function.name, ": DIRECTORY PATH INDICATED IN THE lib.path ARGUMENT DOES NOT EXISTS:\n", paste(lib.path, collapse = "\n"))**

**text.check <- c(text.check, tempo.cat)**

**arg.check <- c(arg.check, TRUE)**

**}**

**}**

**}**

**if( ! is.null(R.lib.path)){**

**tempo <- fun\_check(data = R.lib.path, class = "character", fun.name = function.name) ; eval(ee)**

**if(tempo$problem == FALSE){**

**if( ! all(dir.exists(R.lib.path))){** *# separation to avoid the problem of tempo$problem == FALSE and R.lib.path == NA*

**tempo.cat <- paste0("ERROR IN ", function.name, ": DIRECTORY PATH INDICATED IN THE R.lib.path ARGUMENT DOES NOT EXISTS:\n", paste(R.lib.path, collapse = "\n"))**

**text.check <- c(text.check, tempo.cat)**

**arg.check <- c(arg.check, TRUE)**

**}**

**}**

**}**

**if(any(arg.check) == TRUE){**

**stop(paste0("\n\n================\n\n", paste(text.check[arg.check], collapse = "\n"), "\n\n================\n\n"), call. = FALSE)** *#*

**}**

*# source("C:/Users/Gael/Documents/Git\_versions\_to\_use/debugging\_tools\_for\_r\_dev-v1.2/r\_debugging\_tools-v1.2.R") ; eval(parse(text = str\_basic\_arg\_check\_dev)) ; eval(parse(text = str\_arg\_check\_with\_fun\_check\_dev)) # activate this line and use the function (with no arguments left as NULL) to check arguments status and if they have been checked using fun\_check()*

*# end argument checking*

*# package checking*

**fun\_pack(req.package = "reticulate", lib.path = R.lib.path)**

*# end package checking*

*# main code*

**if(is.null(python.exec.path)){**

**python.exec.path <- reticulate::py\_run\_string("**

**import sys ;**

**path\_lib = sys.path**

**")** *# python string*

**python.exec.path <- python.exec.path$path\_lib**

**}**

**if(is.null(lib.path)){**

**lib.path <- reticulate::py\_run\_string("**

**import sys ;**

**path\_lib = sys.path**

**")** *# python string*

**lib.path <- lib.path$path\_lib**

**}**

**reticulate::use\_python(Sys.which(python.exec.path), required = TRUE)** *# required to avoid the use of erratic python exec by reticulate::import\_from\_path()*

**for(i1 in 1:length(req.package)){**

**tempo.try <- vector("list", length = length(lib.path))**

**for(i2 in 1:length(lib.path)){**

**tempo.try[[i2]] <- suppressWarnings(try(reticulate::import\_from\_path(req.package[i1], path = lib.path[i2]), silent = TRUE))**

**tempo.try[[i2]] <- suppressWarnings(try(reticulate::import\_from\_path(req.package[i1], path = lib.path[i2]), silent = TRUE))** *# done twice to avoid the error message about flushing present the first time but not the second time. see https://stackoverflow.com/questions/57357001/reticulate-1-13-error-in-sysstdoutflush-attempt-to-apply-non-function*

**}**

**if(all(sapply(tempo.try, FUN = grepl, pattern = "[Ee]rror"))){**

**print(tempo.try)**

**tempo.cat <- paste0("ERROR IN ", function.name, ": PACKAGE ", req.package[i1], " MUST BE INSTALLED IN THE MENTIONNED DIRECTORY:\n", paste(lib.path, collapse = "\n"))**

**stop(paste0("\n\n================\n\n", tempo.cat, "\n\n================\n\n"), call. = FALSE)** *# == in stop() to be able to add several messages between ==*

**}** *# else{*

*# suppressMessages(suppressWarnings(suppressPackageStartupMessages(assign(req.package[i1], reticulate::import(req.package[i1]))))) # not required because try() already evaluates*

*# }*

**}**

**}**

################ Print / Exporting results (text & tables)

######## fun\_report() #### print string or data object into output file

**# Check OK: clear to go Apollo**

**fun\_report <- function(data, 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 PACKAGES*

*# utils*

*# REQUIRED FUNCTIONS FROM CUTE\_LITTLE\_R\_FUNCTION*

*# fun\_check()*

*# ARGUMENTS*

*# data: object to print in the output file. If NULL, nothing is done, with no warning*

*# 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\_report()*

*# fun\_report(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(utils::find("fun\_check", mode = "function")) == 0){**

**tempo.cat <- paste0("ERROR IN ", function.name, ": REQUIRED fun\_check() FUNCTION IS MISSING IN THE R ENVIRONMENT")**

**stop(paste0("\n\n================\n\n", tempo.cat, "\n\n================\n\n"), call. = FALSE)** *# == in stop() to be able to add several messages between ==*

**}**

*# end required function checking*

*# argument checking*

**arg.check <- NULL** *#*

**text.check <- NULL** *#*

**checked.arg.names <- NULL** *# for function debbuging: used by r\_debugging\_tools*

**ee <- expression(arg.check <- c(arg.check, tempo$problem) , text.check <- c(text.check, tempo$text) , checked.arg.names <- c(checked.arg.names, tempo$fun.name))**

**tempo <- fun\_check(data = output, class = "character", length = 1, fun.name = function.name) ; eval(ee)**

**if(tempo$problem == FALSE & output == ""){**

**tempo.cat <- paste0("ERROR IN ", function.name, ": output ARGUMENT AS \"\" DOES NOT CORRESPOND TO A VALID FILE NAME")**

**text.check <- c(text.check, tempo.cat)**

**arg.check <- c(arg.check, TRUE)**

**}**

**tempo <- fun\_check(data = path, class = "vector", mode = "character", fun.name = function.name) ; eval(ee)**

**if(tempo$problem == FALSE){**

**if( ! all(dir.exists(path))){** *# separation to avoid the problem of tempo$problem == FALSE and lib.path == NA*

**tempo.cat <- paste0("ERROR IN ", function.name, ": path ARGUMENT DOES NOT CORRESPOND TO EXISTING DIRECTORY\n", paste(path, collapse = "\n"))**

**text.check <- c(text.check, tempo.cat)**

**arg.check <- c(arg.check, TRUE)**

**}**

**}**

**tempo <- fun\_check(data = no.overwrite, class = "logical", length = 1, fun.name = function.name) ; eval(ee)**

**tempo <- fun\_check(data = rownames.kept, class = "logical", length = 1, fun.name = function.name) ; eval(ee)**

**tempo <- fun\_check(data = vector.cat, class = "logical", length = 1, fun.name = function.name) ; eval(ee)**

**tempo <- fun\_check(data = noquote, class = "logical", length = 1, fun.name = function.name) ; eval(ee)**

**tempo <- fun\_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(paste0("\n\n================\n\n", paste(text.check[arg.check], collapse = "\n"), "\n\n================\n\n"), call. = FALSE)** *#*

**}**

*# end 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)) ; eval(parse(text = str\_arg\_check\_with\_fun\_check\_dev)) # activate this line and use the function (with no arguments left as NULL) to check arguments status and if they have been checked using fun\_check()*

*# 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("ERROR 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 = " "))*

*# stop(paste0("\n\n================\n\n", tempo.cat, "\n\n================\n\n"), call. = FALSE) # == in stop() to be able to add several messages between ==*

*# }*

*# end argument checking*

*# main code*

**if( ! is.null(data)){**

**if(all(class(data) == "data.frame") | all(class(data) == "table") | all(class(data) %in% c("matrix", "array"))){** *# before R4.0.0, it was 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 (because 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) == "table") | all(class(data) %in% c("matrix", "array")))){** *# before R4.0.0, it was & all(class(data) %in% c("matrix", "table"))*

**rownames(data) <- rep("", nrow(data))** *# identical row names allowed in matrices and tables*

**}**

**if(noquote == TRUE){**

**utils::capture.output(noquote(data), file=paste0(path, "/", output), append = no.overwrite)**

**}else{**

**utils::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 if(all(mode(data) == "character")){** *# characters (array, list, factor or vector with vector.cat = FALSE)*

**if(noquote == TRUE){**

**utils::capture.output(noquote(data), file=paste0(path, "/", output), append = no.overwrite)**

**}else{**

**utils::capture.output(data, file=paste0(path, "/", output), append = no.overwrite)**

**}**

**}else{** *# other object (S4 for instance, which do not like noquote()*

**utils::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*

**}**

**}**

######## fun\_get\_message() #### return error/warning/other messages of an expression (that can be exported)

**# Check OK: clear to go Apollo**

**fun\_get\_message <- function(data, kind = "error", header = TRUE, print.no = FALSE, text = NULL, env = NULL){**

*# AIM*

*# evaluate an instruction written between "" and return the first of the error, or warning or standard (non error non warning) messages if ever exist*

*# using argument print.no = FALSE, return NULL if no message, which is convenient in some cases*

*# WARNING*

*# Only the first message is returned*

*# Always use the env argument when fun\_get\_message() is used inside functions*

*# REQUIRED FUNCTIONS FROM CUTE\_LITTLE\_R\_FUNCTION*

*# fun\_check()*

*# ARGUMENTS*

*# data: character string to evaluate*

*# kind: character string. Either "error" to get error messages, or "warning" to get warning messages, or "message" to get non error and non warning messages*

*# header: logical. Add a header in the returned message?*

*# print.no: logical. Print a message saying that no message reported?*

*# text: character string added to the output message (even if no message exists and print.no is TRUE). Inactivated if header is FALSE*

*# env: the name of an existing environment. NULL if not required*

*# RETURN*

*# the message or NULL if no message and print.no is FALSE*

*# EXAMPLES*

*# fun\_get\_message(data = "wilcox.test(c(1,1,3), c(1, 2, 4), paired = TRUE)", kind = "error", print.no = TRUE, text = "IN A")*

*# fun\_get\_message(data = "wilcox.test(c(1,1,3), c(1, 2, 4), paired = TRUE)", kind = "warning", print.no = TRUE, text = "IN A")*

*# fun\_get\_message(data = "wilcox.test(c(1,1,3), c(1, 2, 4), paired = TRUE)", kind = "message", print.no = TRUE, text = "IN A")*

*# fun\_get\_message(data = "wilcox.test()", kind = "error", print.no = TRUE, text = "IN A")*

*# fun\_get\_message(data = "sum(1)", kind = "error", print.no = TRUE, text = "IN A")*

*# fun\_get\_message(data = "message('ahah')", kind = "error", print.no = TRUE, text = "IN A")*

*# fun\_get\_message(data = "message('ahah')", kind = "message", print.no = TRUE, text = "IN A")*

*# fun\_get\_message(data = "ggplot2::ggplot(data = data.frame(X = 1:10), mapping = ggplot2::aes(x = X)) + ggplot2::geom\_histogram()", kind = "message", print.no = TRUE, text = "IN FUNCTION 1")*

*# set.seed(1) ; obs1 <- data.frame(Time = c(rnorm(10), rnorm(10) + 2), Group1 = rep(c("G", "H"), each = 10)) ; fun\_get\_message(data = 'fun\_gg\_boxplot(data = obs1, y = "Time", categ = "Group1")', kind = "message", print.no = TRUE, text = "IN FUNCTION 1")*

*# DEBUGGING*

*# data = "wilcox.test(c(1,1,3), c(1, 2, 4), paired = TRUE)" ; kind = "warning" ; header = TRUE ; print.no = FALSE ; text = NULL ; env = NULL # for function debugging*

*# data = "sum(1)" ; kind = "warning" ; header = TRUE ; print.no = FALSE ; text = NULL ; env = NULL # for function debugging*

*# set.seed(1) ; obs1 <- data.frame(Time = c(rnorm(10), rnorm(10) + 2), Group1 = rep(c("G", "H"), each = 10)) ; data = 'fun\_gg\_boxplot(data1 = obs1, y = "Time", categ = "Group1")' ; kind = "warning" ; header = TRUE ; print.no = FALSE ; text = NULL ; env = NULL # for function debugging*

*# data = "message('ahah')" ; kind = "error" ; header = TRUE ; print.no = TRUE ; text = "IN A" ; env = NULL*

*# data = 'ggplot2::ggplot(data = data.frame(X = "a"), mapping = ggplot2::aes(x = X)) + ggplot2::geom\_histogram()' ; kind = "message" ; header = TRUE ; print.no = FALSE ; text = NULL # for function debugging*

*# data = 'ggplot2::ggplot(data = data.frame(X = "a"), mapping = ggplot2::aes(x = X)) + ggplot2::geom\_histogram()' ; kind = "warning" ; header = TRUE ; print.no = FALSE ; text = NULL # for function debugging*

*# data = "emmeans::emmeans(object = emm.rg, specs = contrast.var)" ; kind = "message" ; header = TRUE ; print.no = FALSE ; text = NULL ; env = 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(utils::find("fun\_check", mode = "function")) == 0){**

**tempo.cat <- paste0("ERROR IN ", function.name, ": REQUIRED fun\_check() FUNCTION IS MISSING IN THE R ENVIRONMENT")**

**stop(paste0("\n\n================\n\n", tempo.cat, "\n\n================\n\n"), call. = FALSE)** *# == in stop() to be able to add several messages between ==*

**}**

*# end required function checking*

*# no need to use reserved words to avoid bugs, because it is local, and exists("tempo.warning", inherit = FALSE), never use the scope*

*# argument checking*

*# argument checking with fun\_check()*

**arg.check <- NULL** *#*

**text.check <- NULL** *#*

**checked.arg.names <- NULL** *# for function debbuging: used by r\_debugging\_tools*

**ee <- expression(arg.check <- c(arg.check, tempo$problem) , text.check <- c(text.check, tempo$text) , checked.arg.names <- c(checked.arg.names, tempo$fun.name))**

**tempo <- fun\_check(data = data, class = "character", length = 1, fun.name = function.name) ; eval(ee)**

**tempo <- fun\_check(data = kind, options = c("error", "warning", "message"), length = 1, fun.name = function.name) ; eval(ee)**

**tempo <- fun\_check(data = print.no, class = "logical", length = 1, fun.name = function.name) ; eval(ee)**

**tempo <- fun\_check(data = header, class = "logical", length = 1, fun.name = function.name) ; eval(ee)**

**if( ! is.null(text)){**

**tempo <- fun\_check(data = text, class = "character", length = 1, fun.name = function.name) ; eval(ee)**

**}**

**if( ! is.null(env)){**

**tempo <- fun\_check(data = env, class = "environment", fun.name = function.name) ; eval(ee***) #*

**}**

**if(any(arg.check) == TRUE){**

**stop(paste0("\n\n================\n\n", paste(text.check[arg.check], collapse = "\n"), "\n\n================\n\n"), call. = FALSE)** *#*

**}**

*# end argument checking with fun\_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\_check\_dev)) # activate this line and use the function (with no arguments left as NULL) to check arguments status and if they have been checked using fun\_check()*

*# end argument checking*

*# main code*

**pdf(file = NULL)** *# send plots into a NULL file, no pdf file created*

**window.nb <- dev.cur()**

**invisible(dev.set(window.nb))**

*# last warning cannot be used because suppressWarnings() does not modify last.warning present in the base evironment (created at first warning in a new R session), or warnings() # to reset the warning history : unlockBinding("last.warning", baseenv()) ; assign("last.warning", NULL, envir = baseenv())*

**output <- NULL**

**tempo.error <- try(suppressMessages(suppressWarnings(eval(parse(text = data), envir = if(is.null(env)){parent.frame()}else{env}))), silent = TRUE)** *# get error message, not warning or messages*

**if(any(class(tempo.error) %in% c("gg", "ggplot"))){**

**tempo.error <- try(suppressMessages(suppressWarnings(ggplot2::ggplot\_build(tempo.error))), silent = TRUE)[1]**

**}**

**if(exists("tempo.error", inherit = FALSE) == TRUE){** *# inherit = FALSE avoid the portee lexical and thus the declared word*

**if( ! all(class(tempo.error) == "try-error")){** *# deal with NULL and S4 objects. Old code: ! (all(class(tempo.error) == "try-error") & any(grepl(x = tempo.error, pattern = "^Error|^error|^ERROR"))) but problem with S4 objects. Old code : if((length(tempo.error) > 0 & ! any(grepl(x = tempo.error, pattern = "^Error|^error|^ERROR"))) | (length(tempo.error) == 0) ){ but problem when tempo.error is a list but added this did not work: | ! all(class(tempo.error) == "character")*

**tempo.error <- NULL**

**}**

**}else{**

**tempo.error <- NULL**

**}**

**if(kind == "error" & ! is.null(tempo.error)){** *#*

**if(header == TRUE){**

**tempo.error[1] <- gsub(x = tempo.error[1], pattern = "^Error i|^error i|^ERROR I", replacement = "I")**

**output <- paste0("ERROR MESSAGE REPORTED", ifelse(is.null(text), "", " "), text, ":\n", tempo.error[1])** *#*

**}else{**

**output <- tempo.error[1]** *#*

**}**

**}else if(kind == "error" & is.null(tempo.error) & print.no == TRUE){**

**output <- paste0("NO ERROR MESSAGE REPORTED", ifelse(is.null(text), "", " "), text)**

**}else if(kind != "error" & ( ! is.null(tempo.error)) & print.no == TRUE){**

**output <- paste0("NO ", ifelse(kind == "warning", "WARNING", "STANDARD (NON ERROR AND NON WARNING)"), " MESSAGE BECAUSE OF ERROR MESSAGE REPORTED", ifelse(is.null(text), "", " "), text)**

**}else if(is.null(tempo.error)){**

**fun.warning.capture <- function(expr){**

*# from demo(error.catching) typed in the R console, coming from ?tryCatch*

*# see also http://mazamascience.com/WorkingWithData/?p=912*

*# return a character string or NULL*

**# expr <- wilcox.test.default(c(1, 1, 3), c(1, 2, 4), paired = TRUE)**

**W <- NULL**

**w.handler <- function(w){** *# warning handler*

**W <<- w** *# send to the above env, i.e., the inside of the fun.warning.capture function*

**invokeRestart("muffleWarning")** *# here w.handler() muffles all the warnings. See http://romainfrancois.blog.free.fr/index.php?post/2009/05/20/Disable-specific-warnings to muffle specific warnings and print others*

**}**

**output <- list(**

**value = suppressMessages(withCallingHandlers(tryCatch(expr, error = function(e){e}), warning = w.handler)),** *# BEWARE: w.handler is a function written without (), like in other functions with FUN argument*

**warning = W** *# processed by w.handler()*

**)**

**return(if(is.null(output$warning)){NULL}else{as.character(output$warning)})**

**}**

**tempo.warn <- fun.warning.capture(eval(parse(text = data), envir = if(is.null(env)){parent.frame()}else{env}))**

*# warn.options.ini <- options()$warn ; options(warn = 1) ; tempo.warn <- utils::capture.output({tempo <- suppressMessages(eval(parse(text = data), envir = if(is.null(env)){parent.frame()}else{env}))}, type = "message") ; options(warn = warn.options.ini) # this recover warnings not messages and not errors but does not work in all enviroments*

**tempo.message <- utils::capture.output({**

**tempo <- suppressMessages(suppressWarnings(eval(parse(text = data), envir = if(is.null(env)){parent.frame()}else{env})))**

**if(any(class(tempo) %in% c("gg", "ggplot"))){**

**tempo <- ggplot2::ggplot\_build(tempo)**

**}else{**

**tempo <- suppressWarnings(eval(parse(text = data), envir = if(is.null(env)){parent.frame()}else{env}))**

**}**

**}, type = "message")** *# recover messages not warnings and not errors*

**if(kind == "warning" & ! is.null(tempo.warn)){**

**if(length(tempo.warn) > 0){** *# to avoid character(0)*

**if( ! any(sapply(tempo.warn, FUN = "grepl", pattern = "() FUNCTION:$"))){**

**tempo.warn <- paste(unique(tempo.warn), collapse = "\n")** *# if FALSE, means that the tested data is a special function. If TRUE, means that the data is a standard function. In that case, the output of capture.output() is two strings per warning messages: if several warning messages -> identical first string, which is removed in next messages by unique()*

**}else{**

**tempo.warn <- paste(tempo.warn, collapse = "\n")**

**}**

**if(header == TRUE){**

**if(any(grepl(x = tempo.warn[[1]], pattern = "^simpleWarning i"))){**

**tempo.warn[[1]] <- gsub(x = tempo.warn[[1]], pattern = "^Warning i", replacement = "I")**

**}**

**if(any(grepl(x = tempo.warn[[1]], pattern = "^Warning i"))){**

**tempo.warn[[1]] <- gsub(x = tempo.warn[[1]], pattern = "^Warning i", replacement = "I")**

**}**

**output <- paste0("WARNING MESSAGE REPORTED", ifelse(is.null(text), "", " "), text, ":\n", tempo.warn)** *#*

**}else{**

**output <- tempo.warn** *#*

**}**

**}else{**

**if(print.no == TRUE){**

**output <- paste0("NO WARNING MESSAGE REPORTED", ifelse(is.null(text), "", " "), text)**

**}** *# no need else{} here because output is already NULL at first*

**}**

**}else if(kind == "warning" & is.null(tempo.warn) & print.no == TRUE){**

**output <- paste0("NO WARNING MESSAGE REPORTED", ifelse(is.null(text), "", " "), text)**

**}else if(kind == "message" & exists("tempo.message", inherit = FALSE) == TRUE){** *# inherit = FALSE avoid the portee lexical and thus the declared word*

**if(length(tempo.message) > 0){** *# if something is returned by capture.ouptput() (only in this env) with a length more than 1*

**if(header == TRUE){**

**output <- paste0("STANDARD (NON ERROR AND NON WARNING) MESSAGE REPORTED", ifelse(is.null(text), "", " "), text, ":\n", tempo.message)** *#*

**}else{**

**output <- tempo.message** *#*

**}**

**}else{**

**if(print.no == TRUE){**

**output <- paste0("NO STANDARD (NON ERROR AND NON WARNING) MESSAGE REPORTED", ifelse(is.null(text), "", " "), text)**

**}** *# no need else{} here because output is already NULL at first*

**}**

**}else if(kind == "message" & exists("tempo.message", inherit = FALSE) == FALSE & print.no == TRUE){**

**output <- paste0("NO STANDARD (NON ERROR AND NON WARNING) MESSAGE REPORTED", ifelse(is.null(text), "", " "), text)**

**}** *# no need else{} here because output is already NULL at first*

**}** *# no need else{} here because output is already NULL at first*

**invisible(dev.off(window.nb))** *# end send plots into a NULL file*

**return(output)** *# do not use cat() because the idea is to reuse the message*

**}**