2nd round very low

corr that goes up in loops ? Not possible with pos but possible with values is pos corr decrease & tempo.cor increase -> stop. If pos corr increase before tempo.cor increase -> error

This one is just before I tryed to speed up the code:

**fun\_permut\_consec <- function(data1, data2 = NULL, n = NULL, seed = NULL, count.print = 10, text.print = "", cor.method = "spearman", cor.limit = 0.2, warn.print = FALSE, path.lib = NULL){**

*# AIM*

*# as fun\_permut() except that reorder the elements of the data1 vector by flipping 2 randomly selected consecutive positions either:*

*# 1) n times (when n is precised) or*

*# 2) until the correlation between data1 and data2 decreases down to the cor.limit (0.2 by default). See cor.limit below to deal with negative correlations*

*# Example of consecutive position flipping: ABCD -> BACD -> BADC, etc.*

*# ARGUMENTS*

*# data1: a vector of at least 2 elements. Must be numeric if data2 is specified*

*# data2: a numeric vector of same length as data1*

*# n: number of times "flipping 2 randomly selected consecutive positions". Ignored if data2 is specified*

*# seed: integer number used by set.seed(). 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*

*# count.print: interger value. Print a working progress message every count.print 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 count.print loop*

*# cor.method: correlation method. Either "pearson", "kendall" or "spearman". Ignored if data2 is not specified*

*# cor.limit: a correlation limit (between 0 and 1). 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*

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

*# REQUIRED PACKAGES*

*# lubridate*

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

*# fun\_check()*

*# fun\_pack()*

*# fun\_round()*

*# RETURN*

*# a list containing:*

*# $data: the modified vector*

*# $warnings: 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\_consec(), 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\_consec(), 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\_consec(data1 = LETTERS[1:5], data2 = NULL, n = 100, seed = 1, count.print = 10, text.print = "CPU NB 4")*

*# fun\_permut\_consec(data1 = 101:110, data2 = 21:30, seed = 1, count.print = 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\_consec(data1 = obs1, data2 = obs1, seed = 1, count.print = 10, cor.method = "spearman", cor.limit = 0.2)*

*# fun\_permut\_consec(data1 = 1:1e3, data2 = 1e3:1, seed = 1, count.print = 1e6, text.print = "", cor.method = "spearman", cor.limit = 0.7)*

*# fun\_permut\_consec(data1 = 1:1e2, data2 = 1e2:1, seed = 1, count.print = 1e3, cor.limit = 0.5)*

*# fun\_permut\_consec(data1 = c(0,0,0,0,0), n = 5, data2 = NULL, seed = 1, count.print = 1e3, cor.limit = 0.5)*

*# DEBUGGING*

*# data1 = LETTERS[1:5] ; data2 = NULL ; n = 10 ; seed = NULL ; count.print = 10 ; text.print = "" ; cor.method = "spearman" ; cor.limit = 0.2 ; warn.print = TRUE ; path.lib = NULL*

*# data1 = LETTERS[1:5] ; data2 = NULL ; n = 10 ; seed = 22 ; count.print = 10 ; text.print = "" ; cor.method = "spearman" ; cor.limit = 0.2 ; warn.print = TRUE ; path.lib = NULL*

*# data1 = 101:110 ; data2 = 21:30 ; n = 10 ; seed = 22 ; count.print = 10 ; text.print = "" ; cor.method = "spearman" ; cor.limit = 0.2 ; warn.print = TRUE ; path.lib = NULL*

*# data1 = 1:1e3 ; data2 = 1e3:1 ; n = 20 ; seed = 22 ; count.print = 1e6 ; text.print = "" ; cor.method = "spearman" ; cor.limit = 0.5 ; warn.print = TRUE ; path.lib = NULL*

*# function name*

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

*# end function name*

*# required function checking*

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

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

**stop(tempo.cat)**

**}**

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

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

**stop(tempo.cat)**

**}**

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

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

**stop(tempo.cat)**

**}**

*# end required function checking*

*# argument checking*

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

**checked.arg.names <- NULL** *# for function debbuging*

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

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

**if(tempo$problem == FALSE & length(data1) < 2){**

**tempo.cat <- paste0("\n\n================\n\nERROR IN ", function.name, ": data1 ARGUMENT MUST BE A VECTOR OF MINIMUM LENGTH 2. HERE IT IS: ", length(data1),"\n\n================\n\n")**

**cat(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("\n\n================\n\nERROR IN ", function.name, ": data1 MUST BE A NUMERIC VECTOR IF data2 ARGUMENT IS SPECIFIED\n\n================\n\n")**

**cat(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("\n\n================\n\nERROR IN ", function.name, ": data1 AND data2 MUST BE VECTOR OF SAME LENGTH. HERE IT IS ", length(data1)," AND ", length(data2), "\n\n================\n\n")**

**cat(tempo.cat)**

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

**}**

**}else if(is.null(n)){**

**tempo.cat <- paste0("\n\n================\n\nERROR IN ", function.name, ": n ARGUMENT CANNOT BE NULL IF data2 ARGUMENT IS NULL\n\n================\n\n")**

**cat(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 = count.print, 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(path.lib)){**

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

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

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

**cat(tempo.cat)**

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

**}**

**}**

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

**stop()** *# nothing else because print = TRUE by default in 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*

*# package checking*

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

*# 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*

**warnings <- NULL**

**count <- 0**

**round <- 0**

**BREAK <- FALSE**

**tempo.cor <- 0**

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

**if(length(table(data1)) == 1){**

**tempo.warnings <- paste0("NO PERMUTATION PERFORMED BECAUSE data1 ARGUMENT SEEMS TO BE MADE OF IDENTICAL ELEMENTS: ", names(table(data1)))**

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

**}else{**

**if(count.print > n){**

**count.print <- n**

**}**

*# pos.check <- NULL*

**tempo.count.print <- count.print**

**for(i2 in 1:n){**

**count <- count + 1**

**pos <- sample.int(n = pos.selec.seq.max , size = 1, replace = FALSE)** *# random sample of a position to permute, sample.int samples in 1:pos.selec.seq.max. Or sample(x = pos.selec.seq, size = 1, replace = FALSE) but slower # 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)]**

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

**tempo.count.print <- tempo.count.print + count.print**

**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, " | ")), "FOR LOOP ", i2, " / ", n, " | TIME SPENT: ", tempo.lapse))**

**}**

*# pos.check <- c(pos.check, pos)*

**}**

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

**}**

**}else{**

**if(length(table(data1)) == 1){**

**tempo.warnings <- paste0("NO PERMUTATION PERFORMED BECAUSE data1 ARGUMENT SEEMS TO BE MADE OF IDENTICAL ELEMENTS: ", names(table(data1)))**

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

**tempo.cor <- 1**

**}else if(length(table(data2)) == 1){**

**tempo.warnings <- paste0("NO PERMUTATION PERFORMED BECAUSE data2 ARGUMENT SEEMS TO BE MADE OF IDENTICAL ELEMENTS: ", names(table(data2)))**

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

**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){**

**tempo.warnings <- paste0("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")**

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

**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*

**tempo.warnings <- paste0("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")**

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

**for(i1 in 1:5){** *# done 5 times to be sure of the complete randomness*

**count <- count + 1**

**tempo.pos <- sample(x = tempo.pos, size = length(tempo.pos), replace = FALSE)**

**}**

**}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 = FALSE)** *# 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*

*# 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), " | SMALLEST COR DECREASE: ", fun\_round(smallest.cor.dec, 4)))**

**count.est <- 1e5**

**first.round <- TRUE**

**GOBACK <- FALSE**

**while(tempo.cor > cor.limit + smallest.cor.dec){**

**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(i4 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 1 position*

**for(i5 in 1:count.est){**

**pos2 <- pos[i5]** *# 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[i4] <- 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("\n\n============\n\nERROR IN ", function.name, ": CODE INCONSISTENCY 2\ncor.est.ini: ", cor.est.ini, "\ntempo.cor.est: ", tempo.cor.est, "\n\n============\n\n"))**

**stop(tempo.cat)**

**}**

**cor.dec.per.loop[i4] <- 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("\n\n============\n\nERROR 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, "\n\n============\n\n"))**

**stop(tempo.cat)**

**}else if(loop.nb.est > 1e4){**

**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)))**

**tempo.date.loop <- Sys.time()**

**tempo.time.loop <- as.numeric(tempo.date.loop)**

**count3 <- 0**

**tempo.count.print <- count.print**

**pos <- sample.int(n = pos.selec.seq.max , size = loop.nb.est, replace = TRUE)** *# selection of random positions*

**for(i4 in 1:loop.nb.est){**

**count <- count + 1**

**count3 <- count3 + 1**

**pos2 <- pos[i4] # selection of 1 position**

**tempo.pos[c(pos2 + 1, pos2)] <- tempo.pos[c(pos2, pos2 + 1)]**

**if(count3 == tempo.count.print){**

**tempo.count.print <- tempo.count.print + count.print**

**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) / i4 \* loop.nb.est** *# 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(i4, big.mark=","), " / ", format(loop.nb.est, big.mark=","), " | COUNT: ", format(count, big.mark=","), " | TIME SPENT: ", tempo.lapse, " | EXPECTED END: ", final.exp))**

**}**

**}**

**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)))**

**tempo.date.loop <- Sys.time()**

**tempo.time.loop <- as.numeric(tempo.date.loop)**

**count4 <- 0**

**tempo.count.print <- count.print**

**tempo.cor.loop <- tempo.cor**

**while(tempo.cor > cor.limit){**

**count <- count + 1**

**count4 <- count4 + 1**

**pos <- sample.int(n = pos.selec.seq.max , size = 1, replace = TRUE)** *# selection of random positions*

**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))**

**if(count4 == tempo.count.print){**

**tempo.count.print <- tempo.count.print + count.print**

**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)** *# 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(count4, 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(warning)){**

**warning(warning)**

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

**}**

**output <- list(data = data1[tempo.pos], warnings = warnings, cor = if(is.null(data2)){cor(ini.pos, tempo.pos, method = "spearman")}else{tempo.cor}, count = count)**

**return(output)**

**}**

This one is the non consecutive random permutation

######## fun\_permut() #### progressively breaks a vector order

**fun\_permut <- function(data1, data2 = NULL, n = NULL, seed = NULL, count.print = 10, text.print = "", cor.method = "spearman", cor.limit = 0.2, warn.print = FALSE, path.lib = NULL){**

*# AIM*

*# reorder the elements of the data1 vector by flipping 2 randomly selected 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 position flipping: ABCD -> DBCA -> DACB, etc.*

*# ARGUMENTS*

*# data1: a vector of at least 2 elements. Must be numeric if data2 is specified*

*# data2: a numeric vector of same length as data1*

*# n: number of times "flipping 2 randomly selected 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*

*# count.print: interger value. Print a working progress message every count.print 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 count.print loop*

*# cor.method: correlation method. Either "pearson", "kendall" or "spearman". Ignored if data2 is not specified*

*# cor.limit: a correlation limit (between 0 and 1). 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*

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

*# REQUIRED PACKAGES*

*# lubridate*

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

*# fun\_check()*

*# fun\_pack()*

*# fun\_round()*

*# RETURN*

*# a list containing:*

*# $data: the modified vector*

*# $warnings: 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 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 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, count.print = 10, text.print = "CPU NB 4")*

*# fun\_permut(data1 = 101:110, data2 = 21:30, seed = 1, count.print = 10, 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, count.print = 10, cor.method = "spearman", cor.limit = 0.2)*

*# fun\_permut(data1 = 1:1e3, data2 = 1e3:1, n = 20, seed = 1, count.print = 1e6, text.print = "", cor.method = "spearman", cor.limit = 0.7)*

*# fun\_permut(data1 = 1:1e2, data2 = 1e2:1, seed = 1, count.print = 1e3, cor.limit = 0.5)*

*# fun\_permut(data1 = c(0,0,0,0,0), n = 5, data2 = NULL, seed = 1, count.print = 1e3, cor.limit = 0.5)*

*# DEBUGGING*

*# data1 = LETTERS[1:5] ; data2 = NULL ; n = 100 ; seed = NULL ; count.print = 10 ; text.print = "" ; cor.method = "spearman" ; cor.limit = 0.2 ; warn.print = TRUE ; path.lib = NULL*

*# data1 = LETTERS[1:5] ; data2 = NULL ; n = 10 ; seed = 22 ; count.print = 10 ; text.print = "" ; cor.method = "spearman" ; cor.limit = 0.2 ; warn.print = TRUE ; path.lib = NULL*

*# data1 = 101:110 ; data2 = 21:30 ; n = 10 ; seed = 22 ; count.print = 10 ; text.print = "" ; cor.method = "spearman" ; cor.limit = 0.2 ; warn.print = TRUE ; path.lib = NULL*

*# data1 = 1:1e3 ; data2 = 1e3:1 ; n = 20 ; seed = 22 ; count.print = 1e6 ; text.print = "" ; cor.method = "spearman" ; cor.limit = 0.7 ; warn.print = TRUE ; path.lib = NULL*

*# function name*

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

*# end function name*

*# required function checking*

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

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

**stop(tempo.cat)**

**}**

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

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

**stop(tempo.cat)**

**}**

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

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

**stop(tempo.cat)**

**}**

*# end required function checking*

*# argument checking*

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

**checked.arg.names <- NULL** *# for function debbuging*

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

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

**if(tempo$problem == FALSE & length(data1) < 2){**

**tempo.cat <- paste0("\n\n================\n\nERROR IN ", function.name, ": data1 ARGUMENT MUST BE A VECTOR OF MINIMUM LENGTH 2. HERE IT IS: ", length(data1),"\n\n================\n\n")**

**cat(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("\n\n================\n\nERROR IN ", function.name, ": data1 MUST BE A NUMERIC VECTOR IF data2 ARGUMENT IS SPECIFIED\n\n================\n\n")**

**cat(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("\n\n================\n\nERROR IN ", function.name, ": data1 AND data2 MUST BE VECTOR OF SAME LENGTH. HERE IT IS ", length(data1)," AND ", length(data2), "\n\n================\n\n")**

**cat(tempo.cat)**

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

**}**

**}else if(is.null(n)){**

**tempo.cat <- paste0("\n\n================\n\nERROR IN ", function.name, ": n ARGUMENT CANNOT BE NULL IF data2 ARGUMENT IS NULL\n\n================\n\n")**

**cat(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 = count.print, 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(path.lib)){**

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

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

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

**cat(tempo.cat)**

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

**}**

**}**

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

**stop()** *# nothing else because print = TRUE by default in 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*

*# package checking*

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

*# end package checking*

*# main code*

**warning <- NULL**

*# 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.seq.max <- max(ini.pos, na.rm = TRUE)** *# max position (used by sample.int() function)*

**warnings <- NULL**

*# variable allocation before the loops to save time*

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

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

**final.loop <- tempo.time**

**final.exp <- Sys.time()**

**pos <- ini.pos[1:2]** *# 2 position in initial position, without the last because always up permutation (pos -> pos+1 & pos+1 -> pos)*

**count <- 0**

**tempo.cor <- 0**

*# end variable allocation before the loops to save time*

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

**if(length(table(data1)) == 1){**

**tempo.warnings <- paste0("NO PERMUTATION PERFORMED BECAUSE data1 ARGUMENT SEEMS TO BE MADE OF IDENTICAL ELEMENTS: ", names(table(data1)))**

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

**}else{**

**if(count.print > n){**

**count.print <- n**

**}**

*# pos.check <- NULL*

**cat(paste0("\n", ifelse(text.print == "", "", paste0(text.print, " | ")), "FOR LOOP OF ", n, " LOOPS INITIATED | LOOP COUNT: ", format(count, big.mark=",")))**

**count.print.loop <- logical(length = n)**

**count.print.loop[(1:trunc(n / count.print) \* count.print)] <- TRUE** *# counter to speedup*

**tempo.date.loop <- Sys.time()**

**tempo.time.loop <- as.numeric(tempo.date.loop)**

**for(i2 in 1:n){**

**pos <- sample.int(n = pos.selec.seq.max , size = 1, replace = FALSE)** *# random sample of a position to permute, sample.int samples in 1:pos.selec.seq.max. Or sample(x = pos.selec.seq, size = 1, replace = FALSE) but slower # 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)]**

**if(count.print.loop[i2]){**

**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) / i2 \* n**

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

**cat(paste0("\n", ifelse(text.print == "", "", paste0(text.print, " | ")), "FOR LOOP ", i2, " / ", n, " | TIME SPENT: ", tempo.lapse, " | EXPECTED END: ", final.exp))**

**}**

*# pos.check <- c(pos.check, pos)*

**}**

**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){**

**tempo.warnings <- paste0("NO PERMUTATION PERFORMED BECAUSE data1 ARGUMENT SEEMS TO BE MADE OF IDENTICAL ELEMENTS: ", names(table(data1)))**

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

**tempo.cor <- 1**

**}else if(length(table(data2)) == 1){**

**tempo.warnings <- paste0("NO PERMUTATION PERFORMED BECAUSE data2 ARGUMENT SEEMS TO BE MADE OF IDENTICAL ELEMENTS: ", names(table(data2)))**

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

**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){**

**tempo.warnings <- paste0("INITIAL ", toupper(cor.method), " CORRELATION BETWEEN data1 AND data2 HAS BEEN DETECTED AS NEGATIVE: ", tempo.cor, ". THE cor.limit PARAMETER WILL BE SWITCHED TO THE NEGATIVE EQUIVALENT: ", -cor.limit)**

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

**neg.cor <- TRUE**

**tempo.cor <- abs(tempo.cor)**

**}**

**if(tempo.cor < cor.limit){** *# randomize directly all the position to be close to correlation zero*

**tempo.warnings <- paste0("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")**

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

**cat(paste0("\n", ifelse(text.print == "", "", paste0(text.print, " | ")), "FOR LOOP TO TOTALLY RANDOMIZE data1 BECAUSE TEMPO CORRELATION BELOW CORRELATION LIMIT | LOOP COUNT: ", format(count, big.mark=","), " | CORRELATION LIMIT: ", fun\_round(cor.limit, 4), " | ABS TEMPO CORRELATION: ", fun\_round(tempo.cor, 4)))**

**for(i1 in 1:5){** *# done 5 times to be sure of the complete randomness*

**count <- count + 1**

**tempo.pos <- sample(x = tempo.pos, size = length(tempo.pos), replace = FALSE)**

**}**

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

**}else{**

**cat(paste0("\n", ifelse(text.print == "", "", paste0(text.print, " | ")), "WHILE LOOP INITIATED | LOOP COUNT: ", format(count, big.mark=","), " | CORRELATION LIMIT: ", fun\_round(cor.limit, 4), " | ABS TEMPO CORRELATION: ", fun\_round(tempo.cor, 4)))**

**while(tempo.cor >= cor.limit){**

**count <- count + 1**

**pos <- sample.int(n = pos.seq.max, size = 2, replace = FALSE)** *# selection of 1 position*

**tempo.pos[pos] <- tempo.pos[rev(pos)]**

**tempo.cor <- abs(cor(x = data1[tempo.pos], y = data2, use = "pairwise.complete.obs", method = cor.method))**

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

**tempo.count.print <- tempo.count.print + count.print**

**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 LOOP ", format(count, 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))**

**}**

**}**

**cat(paste0("\n", ifelse(text.print == "", "", paste0(text.print, " | ")), "WHILE LOOP ENDED | LOOP COUNT: ", format(count, big.mark=","), " | CORRELATION LIMIT: ", fun\_round(cor.limit, 4), " | ABS TEMPO CORRELATION: ", fun\_round(tempo.cor, 4)))**

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

**}**

**tempo.cor <- ifelse(neg.cor == TRUE, -tempo.cor, tempo.cor)**

**}**

**}**

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

**warning(warning)**

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

**}**

**output <- list(data = data1[tempo.pos], warnings = warnings, cor = if(is.null(data2)){cor(ini.pos, tempo.pos, method = "spearman")}else{tempo.cor}, count = count)**

**return(output)**

**}**

using swap distance and finishing by while loop

**fun\_permut\_consec <- function(data1, data2 = NULL, n = NULL, seed = NULL, count.print = 10, text.print = "", cor.method = "spearman", cor.limit = 0.2, warn.print = FALSE, path.lib = NULL){**

*# AIM*

*# as fun\_permut() except that reorder the elements of the data1 vector by flipping 2 randomly selected consecutive positions either:*

*# 1) n times (when n is precised) or*

*# 2) until the correlation between data1 and data2 decreases down to the cor.limit (0.2 by default). See cor.limit below to deal with negative correlations*

*# Example of consecutive position flipping: ABCD -> BACD -> BADC, etc.*

*# ARGUMENTS*

*# data1: a vector of at least 2 elements. Must be numeric if data2 is specified*

*# data2: a numeric vector of same length as data1*

*# n: number of times "flipping 2 randomly selected consecutive positions". Ignored if data2 is specified*

*# seed: integer number used by set.seed(). 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*

*# count.print: interger value. Print a working progress message every count.print 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 count.print loop*

*# cor.method: correlation method. Either "pearson", "kendall" or "spearman". Ignored if data2 is not specified*

*# cor.limit: a correlation limit (between 0 and 1). 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*

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

*# REQUIRED PACKAGES*

*# lubridate*

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

*# fun\_check()*

*# fun\_pack()*

*# fun\_round()*

*# RETURN*

*# a list containing:*

*# $data: the modified vector*

*# $warnings: 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\_consec(), 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\_consec(), 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\_consec(data1 = LETTERS[1:5], data2 = NULL, n = 100, seed = 1, count.print = 10, text.print = "CPU NB 4")*

*# fun\_permut\_consec(data1 = 101:110, data2 = 21:30, seed = 1, count.print = 10, 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\_consec(data1 = obs1, data2 = obs1, seed = 1, count.print = 10, cor.method = "spearman", cor.limit = 0.2)*

*# fun\_permut\_consec(data1 = 1:1e3, data2 = 1e3:1, seed = 1, count.print = 1e6, text.print = "", cor.method = "spearman", cor.limit = 0.7)*

*# fun\_permut\_consec(data1 = 1:1e2, data2 = 1e2:1, seed = 1, count.print = 1e3, cor.limit = 0.5)*

*# fun\_permut\_consec(data1 = c(0,0,0,0,0), n = 5, data2 = NULL, seed = 1, count.print = 1e3, cor.limit = 0.5)*

*# DEBUGGING*

*# data1 = LETTERS[1:5] ; data2 = NULL ; n = 10 ; seed = NULL ; count.print = 10 ; text.print = "" ; cor.method = "spearman" ; cor.limit = 0.2 ; warn.print = TRUE ; path.lib = NULL*

*# data1 = LETTERS[1:5] ; data2 = NULL ; n = 10 ; seed = 22 ; count.print = 10 ; text.print = "" ; cor.method = "spearman" ; cor.limit = 0.2 ; warn.print = TRUE ; path.lib = NULL*

*# data1 = 101:110 ; data2 = 21:30 ; n = 10 ; seed = 22 ; count.print = 10 ; text.print = "" ; cor.method = "spearman" ; cor.limit = 0.2 ; warn.print = TRUE ; path.lib = NULL*

*# data1 = 1:1e3 ; data2 = 1e3:1 ; n = 20 ; seed = 22 ; count.print = 1e6 ; text.print = "" ; cor.method = "spearman" ; cor.limit = 0.5 ; warn.print = TRUE ; path.lib = NULL*

*# function name*

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

*# end function name*

*# required function checking*

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

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

**stop(tempo.cat)**

**}**

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

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

**stop(tempo.cat)**

**}**

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

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

**stop(tempo.cat)**

**}**

*# end required function checking*

*# argument checking*

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

**checked.arg.names <- NULL** *# for function debbuging*

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

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

**if(tempo$problem == FALSE & length(data1) < 2){**

**tempo.cat <- paste0("\n\n================\n\nERROR IN ", function.name, ": data1 ARGUMENT MUST BE A VECTOR OF MINIMUM LENGTH 2. HERE IT IS: ", length(data1),"\n\n================\n\n")**

**cat(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("\n\n================\n\nERROR IN ", function.name, ": data1 MUST BE A NUMERIC VECTOR IF data2 ARGUMENT IS SPECIFIED\n\n================\n\n")**

**cat(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("\n\n================\n\nERROR IN ", function.name, ": data1 AND data2 MUST BE VECTOR OF SAME LENGTH. HERE IT IS ", length(data1)," AND ", length(data2), "\n\n================\n\n")**

**cat(tempo.cat)**

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

**}**

**}else if(is.null(n)){**

**tempo.cat <- paste0("\n\n================\n\nERROR IN ", function.name, ": n ARGUMENT CANNOT BE NULL IF data2 ARGUMENT IS NULL\n\n================\n\n")**

**cat(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 = count.print, 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(path.lib)){**

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

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

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

**cat(tempo.cat)**

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

**}**

**}**

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

**stop()** *# nothing else because print = TRUE by default in 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*

*# package checking*

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

*# 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)** *# max position (used by sample.int() function)*

**warnings <- NULL**

**count <- 0**

**round <- 0**

**BREAK <- FALSE**

**tempo.cor <- 0**

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

**if(length(table(data1)) == 1){**

**tempo.warnings <- paste0("NO PERMUTATION PERFORMED BECAUSE data1 ARGUMENT SEEMS TO BE MADE OF IDENTICAL ELEMENTS: ", names(table(data1)))**

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

**}else{**

**if(tempo.count.print > n){**

**tempo.count.print <- n**

**}**

*# pos.check <- NULL*

**tempo.count.print <- count.print**

**for(i2 in 1:n){**

**count <- count + 1**

**pos <- sample.int(n = (pos.selec.seq.max - 1), size = 1, replace = FALSE)** *# random sample of a position to permute, sample.int samples in 1:pos.selec.seq.max. Or sample(x = pos.selec.seq, size = 1, replace = FALSE) but slower # (pos.selec.seq.max - 1) 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)]**

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

**tempo.count.print <- tempo.count.print + count.print**

**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, " | ")), "FOR LOOP ", i2, " / ", n, " | TIME SPENT: ", tempo.lapse))**

**}**

*# pos.check <- c(pos.check, pos)*

**}**

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

**}**

**}else{**

**if(length(table(data1)) == 1){**

**tempo.warnings <- paste0("NO PERMUTATION PERFORMED BECAUSE data1 ARGUMENT SEEMS TO BE MADE OF IDENTICAL ELEMENTS: ", names(table(data1)))**

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

**tempo.cor <- 1**

**}else if(length(table(data2)) == 1){**

**tempo.warnings <- paste0("NO PERMUTATION PERFORMED BECAUSE data2 ARGUMENT SEEMS TO BE MADE OF IDENTICAL ELEMENTS: ", names(table(data2)))**

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

**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){**

**tempo.warnings <- paste0("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")**

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

**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*

**tempo.warnings <- paste0("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")**

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

**for(i1 in 1:5){** *# done 5 times to be sure of the complete randomness*

**count <- count + 1**

**tempo.pos <- sample(x = tempo.pos, size = length(tempo.pos), replace = FALSE)**

**}**

**}else{**

**count <- count + 1** *# 1 and not 0 because already 1 performed just below*

**count2 <- 1** *# 1 and not 0 because already 1 performed just below*

**pos <- sample.int(n = (pos.selec.seq.max - 1), size = 1, replace = FALSE)** *# selection of 1 position # (pos.selec.seq.max - 1) 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))**

*# going out of tempo.cor == cor.ini*

**cat(paste0("\n", ifelse(text.print == "", "", paste0(text.print, " | ")), "FIRST WHILE LOOP STEP | LOOP COUNT: ", format(count, big.mark=","), " | CORRELATION LIMIT: ", fun\_round(cor.limit, 4), " | ABS TEMPO CORRELATION: ", fun\_round(tempo.cor, 4)))**

**tempo.date.loop <- Sys.time()**

**tempo.time.loop <- as.numeric(tempo.date.loop)**

**tempo.count.print <- count.print**

**while(tempo.cor > (cor.ini - 0.0001)){** *# to be out of equality between tempo.cor and cor.ini at the beginning (only valid for very long vector)*

**count <- count + 1**

**count2 <- count2 + 1**

**pos <- sample.int(n = (pos.selec.seq.max - 1), size = 1, replace = FALSE)** *# selection of 1 position # (pos.selec.seq.max - 1) 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))**

**if(count2 == tempo.count.print){**

**tempo.count.print <- tempo.count.print + count.print**

**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 ", format(count2, 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){**

**tempo.warnings <- paste0("THE FIRST FOR & WHILE LOOP STEPS HAVE BEEN TOO FAR AND SUBSEQUENT LOOP STEPS WILL NOT RUN")**

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

**BREAK <- TRUE** *# means tempo.cor.est is negative, and thus go directly to the end #*

**}**

*# end going out of tempo.cor == cor.ini*

*# estimation of the average correlation decrease per loop on 10000 loops and for loop execution*

**cat(paste0("\n", ifelse(text.print == "", "", paste0(text.print, " | ")), "SECOND WHILE/FOR LOOP STEP USING ESTIMATION OF LOOP NB | LOOP COUNT: ", format(count, big.mark=","), " | CORRELATION LIMIT: ", fun\_round(cor.limit, 4), " | ABS TEMPO CORRELATION: ", fun\_round(tempo.cor, 4)))**

**cor.est <- tempo.cor**

**last.swap.dist <- 0**

**max.swap.dist <- 1 # trunc(log10(max(ini.pos, na.rm = TRUE)))**

**count.est <- 1e5**

**while(tempo.cor > cor.limit & BREAK == FALSE){**

**round <- round + 1**

*# estimation step*

**cor.dec.per.loop <- numeric(length = 5)**

**loop.nb.est <- Inf**

**swap.dist <- 0** *# permutation distance*

**cor.est.ini <- cor.est**

**cor.est <- numeric(length = 5)**

**while (loop.nb.est > 1e7 & swap.dist < max.swap.dist){**

**swap.dist <- swap.dist + 1**

**for(i4 in 1:5){** *# connected to cor.dec.per.loop*

**tempo.pos.est <- tempo.pos**

**for(i5 in 1:count.est){**

**pos <- sample.int(n = (pos.selec.seq.max - swap.dist), size = 1, replace = FALSE)** *# selection of 1 position*

**tempo.pos.est[c(pos + swap.dist, pos)] <- tempo.pos.est[c(pos, pos + swap.dist)]**

**}**

**tempo.cor.est <- abs(cor(x = data1[tempo.pos.est], y = data2, use = "pairwise.complete.obs", method = cor.method))**

**cor.est[i4] <- tempo.cor.est**

**tempo.cor.dec.per.loop <- (cor.est.ini - tempo.cor.est) / count.est** *# correlation decrease per loop*

**if(tempo.cor.dec.per.loop < 0){**

**# BREAK <- TRUE** *# means tempo.cor.est is negative, and thus go directly to the end*

**}else if(is.na(tempo.cor.dec.per.loop) | ! is.finite(tempo.cor.dec.per.loop)){**

**tempo.cat <- (paste0("\n\n============\n\nERROR IN ", function.name, ": CODE INCONSISTENCY 2\ncor.est.ini: ", cor.est.ini, "\ntempo.cor.est: ", tempo.cor.est, "\n\n============\n\n"))**

**stop(tempo.cat)**

**}**

**cor.dec.per.loop[i4] <- 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)**

**if(is.na(loop.nb.est) | ! is.finite(loop.nb.est)){**

**tempo.cat <- (paste0("\n\n============\n\nERROR 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, "\n\n============\n\n"))**

**stop(tempo.cat)**

**}**

**}**

*# end estimation step*

*# loop step*

**if(loop.nb.est < 1e4){**

**BREAK <- TRUE**

**cat(paste0("\n", ifelse(text.print == "", "", paste0(text.print, " | ")), "NO MORE FOR LOOP STEP USED BECAUSE ESTIMATION LOOP TOO LOW: ", format(loop.nb.est, big.mark=",")))**

**}else{**

**last.swap.dist <- swap.dist**

**cat(paste0("\n", ifelse(text.print == "", "", paste0(text.print, " | ")), "INITIAL SETTINGS BEFORE FOR LOOP STEP | ROUND: ", round, " | LOOP NUMBER ESTIMATION: ", format(loop.nb.est, big.mark=","), " | SWAPING DISTANCE: ", swap.dist, " | CORRELATION LIMIT: ", fun\_round(cor.limit, 4), " | ABS TEMPO CORRELATION: ", fun\_round(tempo.cor, 4)))**

**tempo.date.loop <- Sys.time()**

**tempo.time.loop <- as.numeric(tempo.date.loop)**

**count3 <- 0**

**tempo.count.print <- count.print**

**for(i4 in 1:loop.nb.est){**

**count <- count + 1**

**count3 <- count3 + 1**

**pos <- sample.int(n = (pos.selec.seq.max - swap.dist), size = 1, replace = FALSE)** *# selection of 1 position*

**tempo.pos[c(pos + swap.dist, pos)] <- tempo.pos[c(pos, pos + swap.dist)]**

**if(count3 == tempo.count.print){**

**tempo.count.print <- tempo.count.print + count.print**

**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) / i4 \* loop.nb.est** *# 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(i4, big.mark=","), " / ", format(loop.nb.est, big.mark=","), " | COUNT: ", format(count, big.mark=","), " | TIME SPENT: ", tempo.lapse, " | EXPECTED END: ", final.exp))**

**}**

**}**

**}**

*# end loop step*

*# end estimation of the average correlation decrease per loop on 10000 loops and for loop execution*

**tempo.cor <- abs(cor(x = data1[tempo.pos], y = data2, use = "pairwise.complete.obs", method = cor.method))**

**}**

**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, " | ")), "SECOND WHILE/FOR LOOP STEP USING ESTIMATION OF LOOP NB END | LOOP COUNT: ", format(count, big.mark=","), " | NB OF ROUNDS: ", round, " | LAST SWAP DISTANCE USED: ", swap.dist, " | 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){**

**tempo.warnings <- paste0("THE SECOND FOR LOOP ESTIMATION STEP HAS BEEN TOO FAR AND THE THIRD WHILE LOOP STEP HAS NOT FINALLY BEEN USED")**

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

**}**

**tempo.date.loop <- Sys.time()**

**tempo.time.loop <- as.numeric(tempo.date.loop)**

**tempo.cor.loop <- tempo.cor**

**count4 <- 0**

**cat(paste0("\n", ifelse(text.print == "", "", paste0(text.print, " | ")), "THIRD WHILE LOOP STEP | LOOP COUNT: ", format(count, big.mark=","), " | CORRELATION LIMIT: ", fun\_round(cor.limit, 4), " | ABS TEMPO CORRELATION: ", fun\_round(tempo.cor, 4), " | ESTIM OF THE NUMBER OF REMAINING LOOPS: ", format(round(loop.nb.est / swap.dist), big.mark=",")))** *# loop.nb.est / swap.dist considering that correlation decrease is proportional to swap.dist*

**tempo.count.print <- count.print**

**while(tempo.cor > cor.limit){**

**count <- count + 1**

**count4 <- count4 + 1**

**pos <- sample.int(n = (pos.selec.seq.max - 1), size = 1, replace = FALSE)** *# selection of 1 position*

**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))**

**if(count4 == tempo.count.print){**

**tempo.count.print <- tempo.count.print + count.print**

**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)** *# 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, " | ")), "THIRD WHILE LOOP ", format(count4, 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, " | ")), "THIRD WHILE LOOP 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))**

**}**

**tempo.cor <- ifelse(neg.cor == TRUE, -tempo.cor, tempo.cor)**

**}**

**}**

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

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

**warning(warning)**

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

**}**

**output <- list(data = data1[tempo.pos], warnings = warnings, cor = if(is.null(data2)){cor(ini.pos, tempo.pos, method = "spearman")}else{tempo.cor}, count = count)**

**return(output)**

**}**