######## fun\_scale() #### select nice numbers when setting breaks on an axis

**# still a bug see the example below**

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

**fun\_scale <- function(lim, n){**

*# AIM*

*# select nice numbers when setting n breaks on a lim axis range*

*# WARNINGS*

*# increase n if the generate scale if not satisfying*

*# ARGUMENTS*

*# lim: vector of 2 numbers indicating the limit range of the axis*

*# n: desired number of breaks on the axis (integer more than 0)*

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

*# fun\_param\_check()*

*# RETURN*

*# a vector of numbers*

*# EXAMPLES*

*# ymin = 2; ymax = 3.101; n = 9; scale <- fun\_scale(lim = c(ymin, ymax), n = n) ; scale ; par(yaxt = "n", yaxs = "i", las = 1) ; plot(ymin:ymax, ymin:ymax, xlab = "DEFAULT SCALE", ylab = "NEW SCALE") ; par(yaxt = "s") ; axis(side = 2, at = scale)*

*# DEBUGGING*

*# lim = c(2, 3.366081) ; n = 4 # for function debugging*

*# lim = c(2, 3.101) ; n = 9 # for function debugging*

*# function name*

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

*# end function name*

*# required function checking*

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

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

**stop(tempo.cat)**

**}**

*# end required function checking*

*# argument checking*

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

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

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

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

**tempo <- fun\_param\_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 & n == 0){**

**tempo.cat <- paste0("\n\n================\n\nERROR IN ", function.name, ": n ARGUMENT MUST BE A NON NULL AND POSITIVE INTEGER\n\n================\n\n")**

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

**}**

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

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

**}**

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

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

*# end argument checking*

*# main code*

**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){**

**tempo.cat <- (paste0("\n\n============\n\nERROR IN ", function.name, ": THE INTERVAL BETWEEN TWO TICKS OF THE SCALE IS NULL. MODIFY THE lim OR n ARGUMENT\n\n============\n\n"))**

**stop(tempo.cat)**

**}**

**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("\n\n============\n\nERROR IN ", function.name, ": CODE INCONSISTENCY 1\n\n============\n\n"))**

**stop(tempo.cat)**

**}**

**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("\n\n============\n\nERROR IN ", function.name, ": CODE INCONSISTENCY 2\n\n============\n\n"))**

**stop(tempo.cat)**

**}**

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

*# end new interval*

*# centering the new scale*

*# detection of the most central tick*

**tempo.mid <- trunc((mid + (-1:1) \* inter.select) / inter.select) \* inter.select** *# tempo middle tick closest to the middle axis*

**mid.tick1 <- tempo.mid[order(abs(tempo.mid - mid)) == 1]** *# 1st central tick*

**mid.tick2 <- tempo.mid[order(abs(tempo.mid - mid)) == 2]** *# 2nd central tick*

**if(mid.tick1 - mid > 0){**

**mid.tick1.above <- TRUE** *# mid.tick1 is above mid and mid.tick2 is below (both frame mid)*

**}else{**

**mid.tick1.above <- FALSE** *# mid.tick1 is below mid and mid.tick2 is above (both frame mid)*

**}**

*# distance from lim of the i-1 and i+1 tick if i is the central tick (odd n)*

**odd.min.dist1 <- (mid.tick1 - inter.select) - tempo.min**

**odd.max.dist1 <- tempo.max - (mid.tick1 + inter.select)**

**odd.closest.dist1 <- abs(odd.min.dist1 - odd.max.dist1)**

**odd.min.dist2 <- (mid.tick2 - inter.select) - tempo.min**

**odd.max.dist2 <- tempo.max - (mid.tick2 + inter.select)**

**odd.closest.dist2 <- abs(odd.min.dist2 - odd.max.dist2)**

**# if(odd.closest.dist1 < odd.closest.dist2){**

**# final.odd.mid.tick <- mid.tick1**

**# }else{**

**# final.odd.mid.tick <- mid.tick2**

**# }**

*# end distance from lim of the i-1 and i+1 tick if i is the central tick (odd n)*

*# distance from lim of the i-1 and i tick or i and i+1 if i is the central tick (even n)*

**even.min.dist1.1 <- mid.tick1 - tempo.min** *# d.min.i*

**even.max.dist1.1 <- tempo.max - (mid.tick1 + inter.select)** *# d.max.i+1*

**even.closest.dist1.1 <- abs(even.min.dist1.1 - even.max.dist1.1)**

*# removed because it is the symmetric of even.closest.dist1.*

**# even.min.dist1.2 <- (mid.tick1 - inter.select) - tempo.min***# d.min.i-1*

**# even.max.dist1.2 <- tempo.max - mid.tick1***# d.max.i*

**# even.closest.dist1.2 <- abs(even.min.dist1.2 - even.max.dist1.2)**

**even.min.dist2.1 <- mid.tick2 - tempo.min** *# d.min.i*

**even.max.dist2.1 <- tempo.max - (mid.tick2 + inter.select)** *# d.max.i+1*

**even.closest.dist2.1 <- abs(even.min.dist2.1 - even.max.dist2.1)**

**# even.min.dist2.2 <- (mid.tick2 - inter.select) - tempo.min***# d.min.i-1*

**# even.max.dist2.2 <- tempo.max - mid.tick2***# d.max.i*

**# even.closest.dist2.2 <- abs(even.min.dist2.2 - even.max.dist2.2)**

**if(even.closest.dist1.1 < even.closest.dist2.1){**

**final.even.mid.tick <- "mid.tick1"**

**}else{**

**final.even.mid.tick <- "mid.tick2"**

**}**

*# end distance from lim of the i-1 and i tick or i and i+1 if i is the central tick (even n)*

*# end detection of the most central tick*

**if(n == 1){**

**output <- mid.tick1**

**}else if(n == 2){**

**output <- c(mid.tick1, ,mid.tick2)** *# the 2 ticks that frame mid*

**}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.tick1 - (trunc(n / 2):1) \* inter.select, mid.tick1, mid.tick1 + (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*

**if(final.even.mid.tick == "mid.tick1"){**

**if(mid.tick1.above == TRUE){**

**output <- c(mid.tick1 - (trunc(n / 2):1) \* inter.select, mid.tick1, mid.tick1 + (1:(trunc(n / 2) - 1)) \* inter.select)** *# one more tick below*

**}else{**

**output <- c(mid.tick1 - ((trunc(n / 2) - 1):1) \* inter.select, mid.tick1, mid.tick1 + (1:trunc(n / 2)) \* inter.select)** *# one more tick above*

**}**

**}else if(final.even.mid.tick == "mid.tick2"){**

**if(mid.tick1.above == TRUE){**

**output <- c(mid.tick2 - ((trunc(n / 2) - 1):1) \* inter.select, mid.tick2, mid.tick2 + (1:trunc(n / 2)) \* inter.select)** *# one more tick above*

**}else{**

**output <- c(mid.tick2 - (trunc(n / 2):1) \* inter.select, mid.tick2, mid.tick2 + (1:(trunc(n / 2) - 1)) \* inter.select)** *# one more tick below*

**}**

**}else{**

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

**stop(tempo.cat)**

**}**

**}else{**

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

**stop(tempo.cat)**

**}**

*# end centering the new scale*

**if(((min(output) - 2 \* inter.select) >= tempo.min) & ((max(output) + inter.select) > tempo.max)){**

**output <- c(min(output) - inter.select, output[-length(output)])**

**}**

*# 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("\n\n============\n\nERROR IN ", function.name, ": CODE INCONSISTENCY 4\n\n============\n\n"))**

**stop(tempo.cat)**

**}**

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

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

**stop(tempo.cat)**

**}**

*# end last check*

**return(output)**

**}**