Package 'CGManalyzer'

May 7, 2026

Type Package

Title Continuous Glucose Monitoring Data Analyzer

Version 1.3.1

Date 2023-7-10

Author Xiaohua Douglas Zhang [aut, cph],
Dandan Wang [aut],
Zhaozhi Zhang [aut],
Madalena Costa [ctb],
Xinzheng Dong [aut, cre]

Maintainer Xinzheng Dong <dong.xinzheng@foxmail.com>

Imports graphics, stats, utils

Description Contains all of the functions necessary for the complete analysis of a continuous glucose monitoring study and can be applied to data measured by various existing 'CGM' devices such as 'FreeStyle Libre', 'Glutalor', 'Dexcom' and 'Medtronic CGM'. It reads a series of data files, is able to convert various formats of time stamps, can deal with missing values, calculates both regular statistics and nonlinear statistics, and conducts group comparison. It also displays results in a concise format. Also contains two unique features new to 'CGM' analysis: one is the implementation of strictly standard mean difference and the class of effect size; the other is the development of a new type of plot called antenna plot. It corresponds to 'Zhang XD'(2018)<doi:10.1093/bioinformatics/btx826>'s article 'CGManalyzer: an R package for analyzing continuous glucose monitoring studies'.

License MIT + file LICENSE

RoxygenNote 6.1.1

NeedsCompilation yes

Repository CRAN

Date/Publication 2023-07-11 12:00:39 UTC

Contents

CGManalyzer-package 2
antennaPlot.fn 13
boxplotCGM.fn 15

CONGA.fn 16
equalInterval.fn 17
fac2char.fn 18
fixMissing.fn 19
MODD.fn 20
MSEbyC.fn 21
MSEplot.fn 22
pairwiseComparison.fn 24
plotTseries.fn 25
setSPEC.fn 26
ssmdEffect.fn 27
summaryCGM.fn 28
timeSeqConversion.fn 29

Index **31**

`CGManalyzer-package` *Continuous Glucose Monitoring Data Analyzer*

Description

Contains all of the functions necessary for the complete analysis of a continuous glucose monitoring study and can be applied to data measured by various existing 'CGM' devices such as 'FreeStyle Libre', 'Glutalor', 'Dexcom' and 'Medtronic CGM'. It reads a series of data files, is able to convert various formats of time stamps, can deal with missing values, calculates both regular statistics and nonlinear statistics, and conducts group comparison. It also displays results in a concise format. Also contains two unique features new to 'CGM' analysis: one is the implementation of strictly standard mean difference and the class of effect size; the other is the development of a new type of plot called antenna plot. It corresponds to 'Zhang XD'(2018)<doi:10.1093/bioinformatics/btx826>'s article 'CGManalyzer: an R package for analyzing continuous glucose monitoring studies'.

Details

The R package CGManalyzer contains functions for analyzing data from a continuous glucose monitoring (CGM) study. It covers a complete flow of data analysis including reading a series of datasets, obtaining summary statistics of glucose levels, plotting data, transforming time stamp format, fixing missing values, calculating multiscale sample entropy (MSE), conducting pairwise comparison, displaying results using various plots including a new type of plots called an antenna plot, etc.. This package has been developed from our work in directly analyzing data from various CGM devices such as FreeStyle Libre, Glutalor, Dexcom, Medtronic CGM. Thus, this package will greatly facilitate the analysis of various CGM studies.

Author(s)

Xiaohua Douglas Zhang [aut, cph], Dandan Wang [aut], Zhaozhi Zhang [aut], Madalena Costa [ctb], Xinzheng Dong [aut, cre]

Maintainer: Xinzheng Dong <dong.xinzheng@foxmail.com>

References

Zhang XD, Zhang Z, Wang D. 2018. CGManalyzer: an R package for analyzing continuous glucose monitoring studies. Bioinformatics 34(9): 1609-1611 (DOI: 10.1093/bioinformatics/btx826).

Costa M., Goldberger A.L., Peng C.-K. Multiscale entropy analysis of physiologic time series. Phys Rev Lett 2002; 89:062102.

Goldberger AL, Amaral LAN, Glass L, Hausdorff JM, Ivanov PCh, Mark RG, Mietus JE, Moody GB, Peng C-K, Stanley HE. PhysioBank, PhysioToolkit, and PhysioNet: Components of a New Research Resource for Complex Physiologic Signals (2003). Circulation. 101(23):e215-e220.

Examples

```
##############################################################################
# The following are the example in the help file for CGManalyzer, which is also
# the main file for using the Package
##############################################################################

# install from local source pacage
# install.packages(paste0(getwd(), "/CGManalyzer_1.2.tar.gz"), repos = NULL)
# or install from CRAN
# install.packages("CGManalyzer")

library(CGManalyzer)

rm(list = ls())
package.name <- "CGManalyzer"
options(scipen = 999)
mainFolder <- getwd()

##############################################################################
# use example data or your own data
##############################################################################
useExampleData <- TRUE

if (useExampleData) {
  SPEC.name <- "SPECexample.R"
} else{
  ############################################################################
  # TODOs before creating file "00filelist.csv":
  # Create a folder named "data" in working directory
  # and copy files into it manually.
  ############################################################################


  ############################################################################
  # Create a file named "00filelist.csv" in "data" folder with two columns,
  # one for file name and one for group type.
  ############################################################################
  dataFolder <-  file.path(mainFolder, "data")
  listFile <- file.path(dataFolder, "00filelist.csv")
  # create a empty file
  if(!file.exists(listFile)){
```

```
    file.create(listFile)
    # list all files and their types
    dataFiles <- list.files(dataFolder)
    if(length(dataFiles)>1){
      dataType <- rep("H",length(dataFiles))
      fileFrame <- data.frame(dataFiles, dataType)
      # save to 00filelist.csv file
      write.csv(fileFrame,listFile,row.names = FALSE,quote = F)
    }else{
      stop("No data files found in the \"data\" folder!")
    }
  }

  ##############################################################################
  # TODOS after creating file "00filelist.csv":
  # Specify the group type for each subject by changing the second
  # column of 00filelist.csv manually. The default type is "H",
  # which needs to be changed to the actual type manually.
  ##############################################################################

  ##############################################################################
  # specify parameters for different CGM devices
  ##############################################################################
  # Choose one of the SPEC file that fits the CGM device that you are using for your study
  # to use the example data in the package "CGManalyzer", you have to choose "SPECexample.R"
  # if you want to run your own data, you cannot choose "SPECexample.R", instead, you need to choose
  # one of "SPEC.FreestyleLibre.R", "SPEC.Glutalor.R" and "SPEC.Medtronic.R".
  # You may also write your own file following the format in one of them.

  SPEC.name <- "SPEC.FreestyleLibre.R"
  # SPEC.name <- "SPEC.Glutalor.R"
  # SPEC.name <- "SPEC.Medtronic.R"
}

setSPEC.fn(SPEC.name)

##############################################################################
## get summary statistics: number of subjects, minimum, 1st quartile, median, mean,
##             mean, 2nd quartile, maximum, number of NA's, standard deviation, MAD
##############################################################################
if (Summary) {
  summary.arr <-
    summaryCGM.fn(
      dataFolder,
      dataFiles,
      responseName,
      sensorIDs,
      columnNames,
      skip = Skip,
      header = Header,
      comment.char = Comment.char,
      sep = Sep
    )
```

```
  # write.csv(file = "summaryStatistics.sensor.csv",
  #           summary.arr[, , 1])
  print(summary.arr[,,1])
}

#############################################################################
## boxplot the data
#############################################################################
if (Boxplot) {
  # filename <- paste("boxplot.rawData", responseName, "pdf",sep=".")
  # pdf(filename)
  yRange <- c(min(summary.arr[, "Min", responseName], na.rm = TRUE),
              max(summary.arr[, "Max", responseName], na.rm = TRUE))
  boxplotCGM.fn(
    dataFolder,
    dataFiles,
    idxNA,
    responseName,
    sensorIDs,
    columnNames,
    yRange,
    skip = Skip,
    header = Header,
    comment.char = Comment.char,
    sep = Sep,
    cex.axis1 = 1
  )
  # dev.off()
}

#############################################################################################
## main analytic process including quality control, interval adjustment, MODD, CONGA, MSE
## calculation
#############################################################################################

fixMissing <- TRUE
fixMethod <- "skip"
calculateMODD <- TRUE
calculateCONGA <- TRUE
n.CONGA <- 2

# filename <- paste("timeSeriesPlot", responseName, "pdf", sep = ".")
# pdf(filename)
par(mfrow = c(3, 2))
for (iFile in 1:nFile) {
  # iFile <- 1 #iFile <- 2 # iFile <- 19
  print(paste0("Process the ", iFile, "th file: ", dataFiles[iFile]))
  data.df0 <-
    read.table(
      paste(dataFolder, dataFiles[iFile], sep = "/"),
      skip = Skip,
      header = Header,
      comment.char = Comment.char,
```

```
    sep = Sep
  )
if (!Header) {
  data.df0 <- data.df0[, 1:length(columnNames)]
  dimnames(data.df0)[[2]] <- columnNames
}

if (!is.na(idxNA))
  data.df0[data.df0[, responseName] == idxNA, responseName] <- NA
#############################################################################
## convert time :  timeSeqConversion.fn
#############################################################################
for (i in 1:length(timeStamp.column)) {
  if (i == 1) {
    timeStamp.vec <- data.df0[, timeStamp.column[i]]
  } else {
    timeStamp.vec <-
      paste0(timeStamp.vec, " ", data.df0[, timeStamp.column[i]])
  }
}
Time.mat <-
  timeSeqConversion.fn(time.stamp = timeStamp.vec,
                       time.format = time.format,
                       timeUnit = timeUnit)
data.df <-
  data.frame(timeStamp.vec, Time.mat[, 1], data.df0[, responseName])
# This needs to be changed if multiple responses
dimnames(data.df)[[2]] <-
  c("timeStamp", "timeSeries", responseName)
data.df <- data.df[order(data.df[, "timeSeries"]),]

#############################################################################
## derive the data with equal interval:  equalInterval.fn
#############################################################################
xx0 <- data.df[, "timeSeries"]
yy0 <- data.df[, responseName]
a <- table(diff(xx0))
if (length(a) > 1)
{
  print("the time interval is not fixed and thus may need to be adjusted.")
}

dataEqualSpace.mat <-
  equalInterval.fn(x = xx0, y = yy0, Interval = equal.interval)

#############################################################################
## fix missing value:  fixMissing.fn
#############################################################################
#fixMissing = TRUE; fixMethod = "linearInterpolation"
if (fixMissing == TRUE) {
  dataFixNA.mat <- fixMissing.fn(y = dataEqualSpace.mat[, "signal"],
                                 x = dataEqualSpace.mat[, "timeSeries"],
                                 Method = fixMethod)
```

```
}

##############################################################################
## calculate MODD:  MODD.fn
##############################################################################
if (calculateMODD == TRUE) {
  theMODD <-
    MODD.fn(y = dataEqualSpace.mat[, "signal"], Interval = equal.interval)
} else {
  theMODD <- NA
}

##############################################################################
## calculate CONGA:  CONGA.fn
##############################################################################
if (calculateCONGA == TRUE) {
  theCONGA <-
    CONGA.fn(y = dataEqualSpace.mat[, "signal"],
             Interval = equal.interval,
             n = n.CONGA)
} else {
  theCONGA <- NA
}

##############################################################################
# calculate multiscale entropy
##############################################################################
xx1 <- dataFixNA.mat[, 1]
yy1 <- dataFixNA.mat[, 2]
theMSE.mat <-
  MSEbyC.fn(
    yy1,
    scaleMax,
    scaleStep,
    mMin = m,
    mMax = m,
    mStep = 1,
    rMin = r,
    rMax = r,
    I = I
  )

summaryAdj.vec <-
  c(
    "N.total" = length(yy0),
    "N.missing" = sum(is.na(yy0)),
    "Mean" = mean(yy1),
    "SD" = sd(yy1),
    "Median" = median(yy1),
    "MAD" = mad(yy1)
  )
if (iFile == 1) {
  summaryAdj.mat <- summaryAdj.vec
```

```
  MSE.mat <- theMSE.mat[, "SampleEntropy"]
  MODD.CONGA.mat <- c(theMODD,  theCONGA)
} else {
  summaryAdj.mat <- rbind(summaryAdj.mat, summaryAdj.vec)
  MSE.mat <- rbind(MSE.mat, theMSE.mat[, "SampleEntropy"])
  MODD.CONGA.mat <-
    rbind(MODD.CONGA.mat, c(theMODD, theCONGA))
}

#########################################################################
## plot data :  plotTseries.fn
#########################################################################
meanY <- mean(yy0, na.rm = TRUE)
plotTseries.fn(
  x = xx0,
  y = yy0,
  xAt = NA,
  xLab = NA,
  yRange = NA,
  Frame = TRUE,
  xlab = paste("Time in", timeUnit),
  ylab = responseName,
  pch = 1,
  lty = 1,
  col.point = 1,
  col.line = 1,
  cex.point = 0.5,
  lwd = 1
)
lines(range(xx0, na.rm = TRUE), rep(meanY, 2), col = "grey")
title(
  main = paste0(iFile, ":", sensorIDs[iFile], ":", subjectTypes[iFile],
                " - Raw Data"),
  sub = paste0(
    "N.total=",
    length(yy0),
    ", N.noNA=",
    sum(!is.na(yy0)),
    ", Mean=",
    round(meanY, 3),
    ", SD=",
    round(sd(yy0, na.rm =
               TRUE), 3)
  )
)

plotTseries.fn(
  x = xx1,
  y = yy1,
  xAt = NA,
  xLab = NA,
  yRange = NA,
  Frame = TRUE,
```

```
    xlab = paste("Time in", timeUnit),
    ylab = responseName,
    pch = 1,
    lty = 1,
    col.point = 1,
    col.line = 1,
    cex.point = 0.5,
    lwd = 1
  )
  lines(range(data.df[, "timeSeries"], na.rm = TRUE), rep(mean(yy1, na.rm =
                                                                 TRUE), 2),
        col = "grey")
  title(
    main = paste0(iFile, ":", sensorIDs[iFile], ":", subjectTypes[iFile],
                  " - Adjusted Data"),
    sub = paste0(
      "N.total=",
      round(summaryAdj.vec["N.total"], 0),
      ", Entropy=",
      theMSE.mat[1, "SampleEntropy"],
      ", Mean=",
      round(summaryAdj.vec["Mean"], 3),
      ", SD=",
      round(summaryAdj.vec["SD"], 3)
    )
  )
}
dimnames(MSE.mat) <- list(sensorIDs, Scales)
dimnames(MODD.CONGA.mat) <- list(sensorIDs, c("MODD", "CONGA"))
dimnames(summaryAdj.mat)[[1]] <- sensorIDs
# dev.off()

##############################################################################################
# compare mean, median, sample entropy et al by group
# there must be at least 2 different types
# in file "00filelist.csv" by modifying manually
##############################################################################################
# Calculate the major results for group comparison among different disease statuses

Types <- unique(subjectTypes)
Types <- Types[order(Types)]
nType <- length(Types)
nPair <- nType * (nType - 1) / 2

# for average value in each type
resultMean.vec <-
  pairwiseComparison.fn(y = summaryAdj.mat[, "Mean"],
                        INDEX = subjectTypes, na.rm =
                          TRUE)

# for MSE in each scale and each type
for (i in 1:dim(MSE.mat)[2]) {
  theResult.vec <-
```

```
    pairwiseComparison.fn(y = MSE.mat[, i],
                          INDEX = subjectTypes,
                          na.rm = TRUE)
  if (i == 1) {
    pvalSSMD.mat <- theResult.vec
  } else {
    pvalSSMD.mat <- rbind(pvalSSMD.mat, theResult.vec)
  }
}
dimnames(pvalSSMD.mat)[1] <- list(Scales)

# write.csv(file = "MSE.csv", MSE.mat)
# write.csv(file = "pvalSSMD.csv", pvalSSMD.mat)
# write.csv(file = "groupComp.mean.csv", round(resultMean.vec, 5))
# write.csv(file = "groupMeanSD.MSE.csv", round(pvalSSMD.mat[,-(1:(nPair *5))], 5))
# write.csv(file = "groupSSMDpvalue.MSE.csv", pvalSSMD.mat[, 1:(nPair *5)])
print(MSE.mat)
print(pvalSSMD.mat)
print(round(resultMean.vec, 5))
print(round(pvalSSMD.mat[,-(1:(nPair *5))], 5))
print(pvalSSMD.mat[, 1:(nPair *5)])


outNames <- dimnames(pvalSSMD.mat)[[2]]
isSSMD <- substring(outNames, 1, 4) == "SSMD"
SSMD.mean.vec <- resultMean.vec[isSSMD]
SSMD.mat <- as.matrix(pvalSSMD.mat[, isSSMD])
ssmdEffect.mat <-
  matrix(
    NA,
    nrow = nrow(SSMD.mat),
    ncol = ncol(SSMD.mat),
    dimnames = dimnames(SSMD.mat)
  )
for (i in 1:ncol(ssmdEffect.mat)) {
  ssmdEffect.mat[, i] <-
    ssmdEffect.fn(SSMD.mat[, i], criterion = "subType")
}
dimnames(ssmdEffect.mat)[1] <-
  list(paste0("sampleEntropy", substring(Scales + 100, 2, 3)))
# write.csv(file = "groupEffect.csv", data.frame(t(ssmdEffect.mat),
#                                    "glucose" = ssmdEffect.fn(SSMD.mean.vec, criterion =
#                                                                          "subType")
#                       ))
print(data.frame(t(ssmdEffect.mat),
                 "glucose" = ssmdEffect.fn(SSMD.mean.vec, criterion =
                                             "subType")
                      ))

################################################################################
# plot sample entropy by individual and by group
################################################################################
scalesInTime <- Scales * equal.interval
```

```
# filename <- "MSEplot.pdf"
# pdf(filename)
par(mfrow = c(1, 1))
col.vec <- rep(NA, length(subjectTypes))
for (i in 1:nType) {
  col.vec[subjectTypes == Types[i]] <- i
}
MSEplot.fn(
  scalesInTime,
  MSE = t(MSE.mat),
  Name = Types,
  responseName = "glucose",
  timeUnit = "minute",
  byGroup = FALSE,
  MSEsd = NA,
  N = NA,
  stdError = TRUE,
  xRange = NA,
  yRange = NA,
  pch = 1,
  las = 2,
  col = col.vec,
  Position = "topleft",
  cex.legend = 0.0005,
  main = "A: MSE by individual"
)
legend(
  "topleft",
  legend = paste0(Types, "(N=", table(subjectTypes), ")"),
  col = 1:nType,
  cex = 1,
  lty = 1,
  pch = 1
)

outNames <- dimnames(pvalSSMD.mat)[[2]]
MSEmean.mat <- pvalSSMD.mat[, substring(outNames, 1, 4) == "mean"]
MSEsd.mat <- pvalSSMD.mat[, substring(outNames, 1, 2) == "SD"]
N.mat <- pvalSSMD.mat[, substring(outNames, 1, 1) == "N"]
MSEplot.fn(
  scalesInTime,
  MSE = MSEmean.mat,
  Name = Types,
  responseName = "glucose",
  timeUnit = "minute",
  byGroup = TRUE,
  MSEsd = MSEsd.mat,
  N = N.mat,
  stdError = TRUE,
  xRange = NA,
  yRange = NA,
  las = 2,
  col = NA,
```

```
  pch = 1:length(Types),
  Position = "topleft",
  cex.legend = 0.75,
  main = "B: MSE by group"
)
# dev.off()

##############################################################################################
# plot results by pairwise comparison of groups : antenna plot
##############################################################################################

# filename <- "antennaPlot.pdf"
# pdf(filename)
par(mfrow = c(1, 1))
# antenna plot for average glucose level
mDiff.vec <- resultMean.vec[substring(outNames, 1, 5) == "mDiff"]
CIlower.vec <-
  resultMean.vec[substring(outNames, 1, 7) == "CIlower"]
CIupper.vec <-
  resultMean.vec[substring(outNames, 1, 7) == "CIupper"]
pairNames <- gsub("mDiff_", "", names(mDiff.vec), fixed = TRUE)
xRange <-
  range(c(
    range(CIlower.vec, na.rm = TRUE),
    0,
    range(CIupper.vec, na.rm = TRUE)
  ))
yRange <- range(c(0, SSMD.mean.vec), na.rm = TRUE)
condt <- !is.na(mDiff.vec) & !is.na(SSMD.mean.vec)
antennaPlot.fn(
  Mean = mDiff.vec[condt],
  SSMD = SSMD.mean.vec[condt],
  Name = pairNames[condt],
  CIlower = CIlower.vec[condt],
  CIupper = CIupper.vec[condt],
  xRange = xRange,
  yRange = yRange,
  col = 1:length(pairNames[condt]),
  pch = 1:length(pairNames[condt]),
  cex = 0.6,
  Position = "bottomright",
  main = "Average Glucose Level"
)

#antenna plots for MSE at each scale
mDiff.mat <-
  as.matrix(pvalSSMD.mat[, substring(outNames, 1, 5) == "mDiff"])
CIlower.mat <-
  as.matrix(pvalSSMD.mat[, substring(outNames, 1, 7) == "CIlower"])
CIupper.mat <-
  as.matrix(pvalSSMD.mat[, substring(outNames, 1, 7) == "CIupper"])
xRange <-
  range(c(
```

```
    range(CIlower.mat, na.rm = TRUE),
    0,
    range(CIupper.mat, na.rm = TRUE)
  ))
yRange <- range(c(0, range(SSMD.mat, na.rm = TRUE)))
for (i in 1:length(Scales)) {
  Main <-
    paste0("Sample entropy at Scale = ",
           Scales[i],
           " (i.e., in ",
           scalesInTime[i],
           " ",
           timeUnit,
           "s)")
  condt <- !is.na(mDiff.mat[i, ]) & !is.na(SSMD.mat[i, ])
  antennaPlot.fn(
    Mean = mDiff.mat[i, condt],
    SSMD = SSMD.mat[i, condt],
    Name = pairNames[condt],
    CIlower = CIlower.mat[i, condt],
    CIupper = CIupper.mat[i, condt],
    xRange = xRange,
    yRange = yRange,
    col = 1:length(pairNames[condt]),
    pch = 1:length(pairNames[condt]),
    cex = 0.6,
    Position = "bottomright",
    main = Main
  )
}
# dev.off()
```

`antennaPlot.fn` *draw an antenna plot*

Description

function to draw an antenna plot

Usage

```
antennaPlot.fn(Mean, SSMD, Name, CIlower, CIupper, xRange = NA, yRange = NA,
col = 1:length(Mean), pch = 1:length(Mean), cex = 1, Position = "topleft", main = "")
```

Arguments

`Mean`	vector for mean difference in a comparison
`SSMD`	vector for strictly standardized mean difference (ssmd) in a comparison
`Name`	vector for name of pairs in a comparison

CIlower	vector for the lower bound of confidence interval
CIupper	vector for the upper bound of confidence interval
xRange	pre-defined range for the x-axis if needed
yRange	pre-defined range for the y-axis if needed
col	vector of colors for pairs in a comparison
pch	vector of point types for pairs in a comparison
cex	cex for the legend
Position	position indicating where to put the legend, such as 'topleft'
main	title name

Details

a function to draw an antenna plot, namely, plot ssmd vs. mean difference with confidence interval.

Value

no return value

Author(s)

Xiaohua Douglas Zhang

References

Zhang XD, Zhang Z, Wang D. 2018. CGManalyzer: an R package for analyzing continuous glucose monitoring studies. Bioinformatics 34(9): 1609-1611 (DOI: 10.1093/bioinformatics/btx826).

Examples

```
library(CGManalyzer)
package.name <- "CGManalyzer"
source( system.file("SPEC", "SPECexample.R", package = package.name) )
scalesInTime <- Scales*equal.interval
pvalSSMD.mat <- read.csv(file=system.file("SPEC", "pvalSSMD.csv", package = package.name),
              row.names=1)
outNames <- dimnames(pvalSSMD.mat)[[2]]
SSMD.mat <- as.matrix( pvalSSMD.mat[, substring(outNames, 1, 4) == "SSMD"] )
mDiff.mat <- as.matrix(pvalSSMD.mat[, substring(outNames, 1, 5) == "mDiff"])
CIlower.mat <- as.matrix(pvalSSMD.mat[, substring(outNames, 1, 7) == "CIlower"])
CIupper.mat <- as.matrix(pvalSSMD.mat[, substring(outNames, 1, 7) == "CIupper"])
pairNames <- gsub("mDiff_", "", dimnames(mDiff.mat)[[2]], fixed=TRUE)
idx = 1:4
xRange <- range( c( range( CIlower.mat[idx,], na.rm=TRUE), 0,
      range( CIupper.mat[idx,], na.rm=TRUE) )  )
yRange <- range( c(0, range( SSMD.mat[idx,], na.rm=TRUE ) ) )
par(mfrow=c(2,2))
for( i in idx ) {
    Main <- paste0("Sample entropy at a scale of ", scalesInTime[i], " ", timeUnit, "s")
    condt <- !is.na(mDiff.mat[i,]) & !is.na(SSMD.mat[i,])
```

```
antennaPlot.fn(Mean=mDiff.mat[i,condt], SSMD=SSMD.mat[i, condt], Name = pairNames[condt],
            CIlower=CIlower.mat[i,condt], CIupper=CIupper.mat[i,condt], xRange=xRange,
          yRange=yRange, col=1:length(pairNames[condt]), pch=1:length(pairNames[condt]),
            cex=0.8, Position = "topleft", main = Main)
}
```

`boxplotCGM.fn` *Draw a boxplot for continuous glucose monitoring data sensor by sensor*

Description

a function to draw a boxplot for continuous glucose monitoring data sensor by sensor

Usage

```
boxplotCGM.fn(dataFolder, dataFiles, idxNA = NA, responseName, sensorIDs,
columnNames = NULL, yRange, skip = 0, header = TRUE, comment.char = "", sep = ",",
cex.axis1 = 0.75)
```

Arguments

`dataFolder`	name for the folder for holding data
`dataFiles`	names of the data files to be read in R
`idxNA`	symbol to represent a missing value, such as NA
`responseName`	name to represent the response to be analyzed, such as 'glucose'
`sensorIDs`	names of sensors or subjects
`columnNames`	names of columns of the data after reading in R
`yRange`	range of y-axis to be drawn in the boxplot
`skip`	number of lines to be skipped in each data file when the data is read in R
`header`	the same meaning as in read.table()
`comment.char`	the same meaning as in read.table()
`sep`	the same meaning as in read.table()
`cex.axis1`	cex for the x-axis

Details

a box plot for the data by each sensor or subject

Value

No value return; draw a boxplot

Author(s)

Xiaohua Douglas Zhang

References

Zhang XD, Zhang Z, Wang D. 2018. CGManalyzer: an R package for analyzing continuous glucose monitoring studies. Bioinformatics 34(9): 1609-1611 (DOI: 10.1093/bioinformatics/btx826).

Examples

```
library(CGManalyzer)
package.name <- "CGManalyzer"
source( system.file("SPEC", "SPECexample.R", package = package.name) )
summary.arr <- summaryCGM.fn(dataFolder, dataFiles, responseName, sensorIDs, columnNames,
            skip=Skip, header=Header, comment.char=Comment.char, sep=Sep)
yRange <- c( min(summary.arr[, "Min",responseName], na.rm=TRUE),
      max(summary.arr[, "Max",responseName], na.rm=TRUE))
boxplotCGM.fn(dataFolder, dataFiles, idxNA, responseName, sensorIDs, columnNames, yRange,
            skip=Skip, header=Header, comment.char=Comment.char, sep=Sep, cex.axis1=1)
```

CONGA.fn *Function to calculate the continuous overlapping net glycemic action (CONGA)*

Description

For each observation after the first n hours of observations, the difference between the current observation and the observation n hours previous was calculated. CONGA is defined as the standard deviation of the differences.

Usage

```
CONGA.fn(y, Interval = 5, n = 2)
```

Arguments

y	measured response, must be evenly spaced in measured time
Interval	number of minutes between two consecutive time points
n	the length of a segment in CONGA

Value

a value of CONGA

Author(s)

Xiaohua Douglas Zhang, Dandan Wang

References

Zhang XD, Zhang Z, Wang D. 2018. CGManalyzer: an R package for analyzing continuous glucose monitoring studies. Bioinformatics 34(9): 1609-1611 (DOI: 10.1093/bioinformatics/btx826).

Examples

```
library(CGManalyzer)
package.name <- "CGManalyzer"
source( system.file("SPEC", "SPECexample.R", package = package.name) )
y = rnorm( 3*24*60/5, mean=5, sd=0.1)
CONGA.fn(y, Interval = 5, n=2)
```

`equalInterval.fn` *Function to derive the data with equal interval*

Description

Function to derive the data with equal interval

Usage

```
equalInterval.fn(x, y, Interval = NA, minGap = 4 * Interval)
```

Arguments

`x`	time sequence
`y`	measured response
`Interval`	interval indicating equal space between two consecutive points
`minGap`	the length of a chain of continuous missing values in which the missing values will not be derived from the neighbor points

Details

Function to derive the data with equal interval for a timeseries

Value

a matrix with equally spaced time sequence and corresponding signal value

Author(s)

Xiaohua Douglas Zhang

References

Zhang XD, Zhang Z, Wang D. 2018. CGManalyzer: an R package for analyzing continuous glucose monitoring studies. Bioinformatics 34(9): 1609-1611 (DOI: 10.1093/bioinformatics/btx826).

Examples

```
data.mat <-
   cbind( "timeSeries"=c(0,  3,  6, 9, 11, 21, 24, 27, 33, 38, 39, 42),
          "signal"=c(3.930, 3.973, 4.005, 4.110, 4.164, 4.165, 4.186,
          4.265, 4.266, 4.357, 4.503, 4.690) )
dataEqualSpace.mat <- equalInterval.fn(x=data.mat[,1], y=data.mat[,2], Interval=3)
data.mat
dataEqualSpace.mat
```

`fac2char.fn` *function to convert a factor to a vector*

Description

function to convert a factor to a vector of characters

Usage

```
fac2char.fn(x)
```

Arguments

x	a factor

Details

function to convert a factor to a vector

Value

a vector of characters

Author(s)

Xiaohua Douglas Zhang

References

Zhang XD, Zhang Z, Wang D. 2018. CGManalyzer: an R package for analyzing continuous glucose monitoring studies. Bioinformatics 34(9): 1609-1611 (DOI: 10.1093/bioinformatics/btx826).

Examples

```
library(CGManalyzer)
package.name <- "CGManalyzer"
source( system.file("SPEC", "SPECexample.R", package = package.name) )
fac2char.fn(dataFileType.df[,1])
```

fixMissing.fn *Function to fix missing values in a vector*

Description

Function to fix missing values in a vector

Usage

```
fixMissing.fn(y, x, Method = c("skip", "linearInterpolation", "loess", "dayCycle"),
OBScycle = 24 * 60 * 60/10)
```

Arguments

y	a vector of data with missing values
x	a vector for a series of consecutive time indices
Method	method options for fixing missing value. "skip": skip all missing values; "loess": use local fitting by loess(); "dayCycle":a missing value is replaced by the mean of the two values one day ahead and one day behind plus the mean of the differences between the two edge points and their corresponding means of the two values one day head and one day behind in a segment with missing values in which the missing value belongs to.
OBScycle	number of observations in a full cycle

Value

a vector of data from 'y' but with missing values fixed

Author(s)

Xiaohua Douglas Zhang

References

Zhang XD, Zhang Z, Wang D. 2018. CGManalyzer: an R package for analyzing continuous glucose monitoring studies. Bioinformatics 34(9): 1609-1611 (DOI: 10.1093/bioinformatics/btx826).

Examples

```
data.mat <-
   cbind( "x"=c(0,  3,  6, 9, 12, 15, 18, 21, 24, 27, 30, 33, 36, 39, 42),
          "signal"=c(3.930, 3.973, 4.005, NA, 4.164, 4.190, 4.205, NA, 4.186,
          4.265, NA, 4.266, 4.357, 4.503, 4.690) )
dataFixNA.mat <- fixMissing.fn( y=data.mat[,2], x=data.mat[,1], Method="linearInterpolation")
data.mat
dataFixNA.mat
```

MODD.fn *Function to calculate the mean of daily differences (MODD)*

Description

Calculates MODD which is the absolute value of the difference between glucose values taken on two consecutive days at the same time was calculated; the MODD is the mean of these differences.

Usage

```
MODD.fn(y, Interval = 5)
```

Arguments

y	measured response, must be evenly spaced in measured time
Interval	number of minutes between two consecutive time points

Value

a value of MODD

Author(s)

Xiaohua Douglas Zhang

References

Zhang XD, Zhang Z, Wang D. 2018. CGManalyzer: an R package for analyzing continuous glucose monitoring studies. Bioinformatics 34(9): 1609-1611 (DOI: 10.1093/bioinformatics/btx826).

Examples

```
library(CGManalyzer)
package.name <- "CGManalyzer"
source( system.file("SPEC", "SPECexample.R", package = package.name) )
y = rnorm( 3*24*60/5, mean=5, sd=0.1)
MODD.fn(y, Interval = 5)
```

`MSEbyC.fn` *call a C function to calculate multiscale entropy*

Description

Function to call a C function to calculate multiscale entropy (MSE) of an equally spaced time series.

Usage

```
MSEbyC.fn(x, scaleMax = 10, scaleStep = 1, mMin = 2, mMax = 2, mStep = 1, rMin = 0.15,
rMax = 0.15, I = 400000)
```

Arguments

`x`	A numeric vector, with data for a regularly spaced time series. No missing value is allowed because the C program is not set up to handle missing value.
`scaleMax`	maximal value of scale factors for coarse graining in the MSE algorithm. The scale factors are a sequence from 1 to a value no more than 'scaleMax' with equal space 'scaleStep'. Scale factors are positive integers that specify bin size for coarse graining: the number of consecutive observations in 'x' that form a bin and are averaged in the first step of the algorithm.
`scaleStep`	see 'scaleMax'
`mMin`	A sequence from 'mMin' to 'mMax' with equal space of 'mStep' that defines the vector of positive integers that give the window size for the entropy calculations in the second step of the algorithm: the number of consecutive _bins_ over which similarity between subsequences is of interest. Typical values in the sequence are 1, 2, or 3.
`mMax`	See 'Min'
`mStep`	See 'Min'
`rMin`	A sequence from 'rMin' to 'rMax' with equal space of 0.05 that defines coefficients for similarity thresholds. Typical values in the sequence are 0.15, 0.2. r*sd(x) must be in the same units as 'x'. Averages in two bins are defined to be similar if they differ by 'r*sd(x)' or less.
`rMax`	See 'rMin'
`I`	the maximal number of points to be used for calculating MSE cFolder: The directory in which .c is held as well as in which temporary files associated with running C are created/removed.

Details

Function to call a C function to calculate multiscale entropy (MSE) of an equally spaced time series.

Value

A data frame with with one row for each combination of 'Scale', 'm' and 'rSD'. Columns are "Scale", "m", "rSD", and "SampEn" (the calculated sample entropy). The data frame will also have an attribute "SD", the standard deviation of 'x'. rSD = r*sd(x)

Author(s)

Xiaohua Douglas Zhang

References

Zhang XD, Zhang Z, Wang D. 2018. CGManalyzer: an R package for analyzing continuous glucose monitoring studies. Bioinformatics 34(9): 1609-1611 (DOI: 10.1093/bioinformatics/btx826).

Examples

```
library(CGManalyzer)
package.name <- "CGManalyzer"
source( system.file("SPEC", "SPECexample.R", package = package.name) )
data.df0 <- read.table(paste(dataFolder, dataFiles[1], sep="/"),
        skip=Skip, header=Header, comment.char=Comment.char, sep=Sep)
if( !Header ) {
data.df0 <- data.df0[, 1:length(columnNames)]
    dimnames(data.df0)[[2]] <-  columnNames
}
if( !is.na(idxNA) ) data.df0[ data.df0[, responseName] == idxNA, responseName] <- NA
for( i in 1:length(timeStamp.column) ) {
if(i==1) { timeStamp.vec <- data.df0[, timeStamp.column[i] ] } else {
 timeStamp.vec <- paste0(timeStamp.vec, " ", data.df0[, timeStamp.column[i] ])
}
}
Time.mat <- timeSeqConversion.fn(time.stamp=timeStamp.vec, time.format=time.format,
            timeUnit=timeUnit)
data.df <- data.frame( timeStamp.vec, Time.mat[,1], data.df0[,responseName] )
dimnames(data.df)[[2]] <- c("timeStamp", "timeSeries", responseName)
data.df <- data.df[ order(data.df[, "timeSeries"]), ]
data.mat <- data.df[, c("timeSeries", responseName)]
data.mat <- data.mat[!is.na(data.mat[,2]), ]
MSE.mat <- MSEbyC.fn(data.mat[,2], scaleMax, scaleStep, mMin=m, mMax=m, mStep=1,
          rMin=r, rMax=r, I=I)
MSE.mat
```

MSEplot.fn *Plot the mean and standard error or standard deviation of multiscale entropy by group*

Description

function to plot the mean and standard error or standard deviation of multiscale entropy by group

Usage

```
MSEplot.fn(Scale, MSE, Name, responseName = NA, timeUnit = "", byGroup = TRUE,
MSEsd = NA, N = NA, stdError = TRUE, xRange = NA, yRange = NA, las = 2, col = NA,
pch = NA, Position = "topleft", cex.legend = 0.75, main = "")
```

Arguments

Scale	a vector for scale
MSE	matrix for entropy if byGroup=FALSE, and otherwise for average entropy value in a group at a scale. In the matrix, the row is for scale and column for individuals or groups.
Name	vector of names for groups
responseName	name to represent the response to be analyzed, such as 'glucose'
timeUnit	the time unit for scale
byGroup	If byGroup = TRUE, multiscale entropy is plotted by groups; otherwise, by individuals
MSEsd	matrix for standard deviation of entropy value in a group at a scale
N	matrix for number of subjects in a group at a scale
stdError	if it is true, the length of a vertical bar represent 2*standard error; otherwise, the length of a vertical bar represent 2*standard deviation
xRange	range for the x-axis
yRange	range for the y-axis
las	las for the y-axis
col	vector for the colors to indicate groups or individuals
pch	vector for the point types to indicate groups or individuals
Position	position for the legend
cex.legend	cex for the legend
main	main title for title()

Details

function to plot the mean and standard error or standard deviation of multiscale entropy by group

Value

No value returned

Author(s)

Xiaohua Douglas Zhang

References

Zhang XD, Zhang Z, Wang D. 2018. CGManalyzer: an R package for analyzing continuous glucose monitoring studies. Bioinformatics 34(9): 1609-1611 (DOI: 10.1093/bioinformatics/btx826).

Examples

```
library(CGManalyzer)
package.name <- "CGManalyzer"
source( system.file("SPEC", "SPECexample.R", package = package.name) )
scalesInTime <- Scales*equal.interval
MSE.mat <- read.csv(file=system.file("SPEC", "MSE.csv", package = package.name), row.names=1)
Types <- unique( subjectTypes )
Types <- Types[order(Types)]
nType <-length(Types)
col.vec <- rep(NA, length(subjectTypes) )
for( i in 1:nType ) { col.vec[ subjectTypes == Types[i] ] <- i }
MSEplot.fn(scalesInTime, MSE=t(MSE.mat), Name=Types, responseName="glucose", timeUnit="minute",
          byGroup=FALSE, MSEsd=NA, N=NA, stdError=TRUE, xRange=NA, yRange=NA,
          pch=rep(1, dim(MSE.mat)[1]),las=2, col=col.vec, Position="topleft",
          cex.legend=0.0005, main="A: MSE by individual")
legend("topleft", legend=paste0(Types, "(N=", table( subjectTypes ), ")"),
      col=1:nType, cex=1, lty=1, pch=1)
```

---

pairwiseComparison.fn *function to calculate mean difference and its confidence interval, SSMD, p-value of t.test for pairwise comparison*

Description

function to calculate mean difference and its confidence interval, SSMD, p-value of t.test for pairwise comparison

Usage

```
pairwiseComparison.fn(y, INDEX, na.rm = TRUE, conf.level = 0.95)
```

Arguments

y	response value
INDEX	vector for group names
na.rm	whether to remove value for calculation
conf.level	confidence level for two-sided t-test

Details

function to calculate mean difference and its confidence interval, SSMD, p-value of t.test for pairwise comparison

Value

a vector for calculated mean difference, its upper and lower bounds of CI, SSMD and pvalue in each pairs of group comparison, along with mean, standard deviation, and sample size in each group

Author(s)

Xiaohua Douglas Zhang

References

Zhang XD, Zhang Z, Wang D. 2018. CGManalyzer: an R package for analyzing continuous glucose monitoring studies. Bioinformatics 34(9): 1609-1611 (DOI: 10.1093/bioinformatics/btx826).

Examples

```
library(CGManalyzer)
package.name <- "CGManalyzer"
source( system.file("SPEC", "SPECexample.R", package = package.name) )
MSE.mat <- read.csv(file=system.file("SPEC", "MSE.csv", package = package.name), row.names=1)
pairwiseComparison.fn(y=MSE.mat[, 1], INDEX=subjectTypes, na.rm=TRUE)
```

---

plotTseries.fn *function to plot time series data*

Description

function to plot time series data

Usage

```
plotTseries.fn(x, y, xAt = NA, xLab = NA, yRange = NA, Frame = TRUE, xlab = "",
ylab = "", pch = 1, lty = 1, col.point = 1, col.line = 1, cex.point = 1, lwd = 1)
```

Arguments

x	time in continuous value such as in seconds or minutes, (e.g. the return from timeSeqConversion.fn)
y	measured response value
xAt	a vector to indicate where the labels in the x-axis are
xLab	a vector to indicate what the labels in the x-axis are
yRange	range for y in the plot
Frame	whether the plot frame should be drawn
xlab	as in plot()
ylab	as in plot()
pch	as in plot()
lty	as in plot()
col.point	the color for the points
col.line	the color for the line
cex.point	cex for the points
lwd	as in plot()

Details

function to plot time series data

Author(s)

Xiaohua Douglas Zhang

References

Zhang XD, Zhang Z, Wang D. 2018. CGManalyzer: an R package for analyzing continuous glucose monitoring studies. Bioinformatics 34(9): 1609-1611 (DOI: 10.1093/bioinformatics/btx826).

Examples

```
library(CGManalyzer)
package.name <- "CGManalyzer"
source( system.file("SPEC", "SPECexample.R", package = package.name) )
data.df0 <- read.table(paste(dataFolder, dataFiles[1], sep="/"),
            skip=Skip, header=Header, comment.char=Comment.char, sep=Sep)
if( !Header ) {
data.df0 <- data.df0[, 1:length(columnNames)]
    dimnames(data.df0)[[2]] <-  columnNames
}
if( !is.na(idxNA) ) data.df0[ data.df0[, responseName] == idxNA, responseName] <- NA
for( i in 1:length(timeStamp.column) ) {
if(i==1) { timeStamp.vec <- data.df0[, timeStamp.column[i] ] } else {
 timeStamp.vec <- paste0(timeStamp.vec, " ", data.df0[, timeStamp.column[i] ])
}
}
Time.mat <- timeSeqConversion.fn(time.stamp=timeStamp.vec, time.format=time.format,
            timeUnit=timeUnit)
data.df <- data.frame( timeStamp.vec, Time.mat[,1], data.df0[,responseName] )
dimnames(data.df)[[2]] <- c("timeStamp", "timeSeries", responseName)
data.df <- data.df[ order(data.df[, "timeSeries"]), ]
plotTseries.fn( x=data.df[, "timeSeries"], y=data.df[, responseName],
xAt=0:14*720, xLab=0:14/2, yRange=NA, Frame=TRUE,
xlab="Time in Days", ylab=responseName, pch=1, lty=1,
col.point=1, col.line=1, cex.point=0.5, lwd=1 )
```

`setSPEC.fn` *Load settings for the selected SPEC parameter.*

Description

A function to load settings for the selected SPEC parameter.

Usage

```
setSPEC.fn(SPEC.name)
```

Arguments

SPEC.name	the SPEC name to load, which is a R script name. one of "SPEC.FreestyleLibre.R", "SPEC.Glutalor.R" and "SPEC.Medtronic.R".

Details

A function to load settings for the selected SPEC parameter. If you want to use the example data in the package "CGManalyzer", you have to choose "SPECexample.R". If you want to run your own data, you cannot choose "SPECexample.R", instead, you need to choose one of "SPEC.FreestyleLibre.R", "SPEC.Glutalor.R" and "SPEC.Medtronic.R".

Value

no value returned

Author(s)

Xiaohua Douglas Zhang

References

Zhang XD, Zhang Z, Wang D. 2018. CGManalyzer: an R package for analyzing continuous glucose monitoring studies. Bioinformatics 34(9): 1609-1611 (DOI: 10.1093/bioinformatics/btx826).

Examples

```
# set SPEC for reading data
package.name <- "CGManalyzer"
options(scipen = 999)
mainFolder <- getwd()
SPEC.name <- "SPECexample.R"
setSPEC.fn(SPEC.name)
```

---

ssmdEffect.fn *function to derive the type of effect size based on SSMD values*

Description

function to derive the type of effect size based on SSMD values

Usage

```
ssmdEffect.fn(ssmd.vec, criterion = c("mainType", "subType"))
```

Arguments

ssmd.vec	a vector for SSMD value
criterion	whether use the criterion for deriving the main effect type or the sub-type

Details

function to derive the type of effect size based on SSMD values

Author(s)

Xiaohua Douglas Zhang

References

Zhang XD, Zhang Z, Wang D. 2018. CGManalyzer: an R package for analyzing continuous glucose monitoring studies. Bioinformatics 34(9): 1609-1611 (DOI: 10.1093/bioinformatics/btx826).

Zhang XHD, 2011. Optimal High-Throughput Screening: Practical Experimental Design and Data Analysis for Genome-scale RNAi Research. Cambridge University Press, Cambridge, UK

Examples

```
SSMD.vec = c(-3.4, -5, 0.198, 0.055, 0.181, 2, 3, 1.5, 6, 0.25)
ssmdEffect.fn(SSMD.vec, criterion="subType")
```

`summaryCGM.fn` *Function to calculate the summary statistics for each subject or sensor: number of subjects or sensors, minimum, 1st quartile, median, mean, 2nd quartile, maximum, standard deviation, MAD*

Description

Function to calculate the summary statistics for each subject or sensor: number of subjects or sensors, minimum, 1st quartile, median, mean, 2nd quartile, maximum, standard deviation, MAD

Usage

```
summaryCGM.fn(dataFolder, dataFiles, responseNames, sensorIDs, columnNames = NULL,
skip = 0, header = TRUE, comment.char = "", sep = ",")
```

Arguments

`dataFolder`	folder directory for holding raw CGM data
`dataFiles`	file names for holding raw CGM data, usually one file for one sensor
`responseNames`	name for the response
`sensorIDs`	ID's for sensors
`columnNames`	column names for the raw data
`skip`	number of lines to be skip in data file when using read.table
`header`	the same as in read.table()
`comment.char`	the same as in read.table()
`sep`	the same as in read.table()

Details

Function to calculate the summary statistics for each subject or sensor: number of subjects or sensors, minimum, 1st quartile, median, mean, 2nd quartile, maximum, standard deviation, MAD

Author(s)

Xiaohua Douglas Zhang

References

Zhang XD, Zhang Z, Wang D. 2018. CGManalyzer: an R package for analyzing continuous glucose monitoring studies. Bioinformatics 34(9): 1609-1611 (DOI: 10.1093/bioinformatics/btx826).

Examples

```
library(CGManalyzer)
package.name <- "CGManalyzer"
source( system.file("SPEC", "SPECexample.R", package = package.name) )
summary.arr <- summaryCGM.fn(dataFolder, dataFiles, responseName, sensorIDs, columnNames,
              skip=Skip, header=Header, comment.char=Comment.char, sep=Sep)
summary.arr[1:6, ,1]
```

---

timeSeqConversion.fn *function to convert a matrix (with columns for year, month, day, minute and/or second) to a time sequence in a unit of minute or second*

Description

function to convert a matrix (with columns for year, month, day, minute and/or second) to a time sequence in a unit of minute or second

Usage

```
timeSeqConversion.fn(time.stamp, time.format = "yyyy:mm:dd:hh:nn", timeUnit = "minute")
```

Arguments

time.stamp	a vector for the time stamp. It can have any format such as "2016:08:11:09:14:00", "11/08/2016 09:14:00" and others. The requirement is simply that the positions for year, month, day, hour, minute, second are fixed and consistent in all data files and "0" before a non-zero number cannot be skipped.
time.format	a string to specify the format in time.stamp in a study, such as "yyyy:mm:dd:hh:nn:ss:ii", "dd/mm/yyyy hh:nn:ss:ii" and others which must have 'y' for year, 'm' for month, 'd' for day, 'h' for hour, 'n' for minute, 's' for second, 'i' for millisecond(one thousandth of a second), each uniquely
timeUnit	minimal time unit in time.stamp. can be 'minute' or 'second'

Details

function to convert a matrix (with columns for year, month, day, minute and/or second) to a time sequence in a unit of minute or second

Value

a matrix with 6 columns for timeseries, year, month, day, hour and minute, respectively

Author(s)

Xiaohua Douglas Zhang

References

Zhang XD, Zhang Z, Wang D. 2018. CGManalyzer: an R package for analyzing continuous glucose monitoring studies. Bioinformatics 34(9): 1609-1611 (DOI: 10.1093/bioinformatics/btx826).

Examples

```
library(CGManalyzer)
package.name <- "CGManalyzer"
source( system.file("SPEC", "SPECexample.R", package = package.name) )
data.df0 <- read.table(paste(dataFolder, dataFiles[1], sep="/"),
            skip=Skip, header=Header, comment.char=Comment.char, sep=Sep)
if( !Header ) {
data.df0 <- data.df0[, 1:length(columnNames)]
    dimnames(data.df0)[[2]] <-  columnNames
}
if( !is.na(idxNA) ) data.df0[ data.df0[, responseName] == idxNA, responseName] <- NA
for( i in 1:length(timeStamp.column) ) {
if(i==1) { timeStamp.vec <- data.df0[, timeStamp.column[i] ] } else {
 timeStamp.vec <- paste0(timeStamp.vec, " ", data.df0[, timeStamp.column[i] ])
}
}
Time.mat <- timeSeqConversion.fn(time.stamp=timeStamp.vec, time.format=time.format,
            timeUnit=timeUnit)
Time.mat[1:6,]
```

Index

∗ **package**
CGManalyzer-package, 2

antennaPlot.fn, 13

boxplotCGM.fn, 15

CGManalyzer *(*CGManalyzer-package*)*, 2
CGManalyzer-package, 2
CONGA.fn, 16

equalInterval.fn, 17

fac2char.fn, 18
fixMissing.fn, 19

MODD.fn, 20
MSEbyC.fn, 21
MSEplot.fn, 22

pairwiseComparison.fn, 24
plotTseries.fn, 25

setSPEC.fn, 26
ssmdEffect.fn, 27
summaryCGM.fn, 28

timeSeqConversion.fn, 29