Package 'RespirAnalyzer'

October 12, 2022

Type Package

Title Analysis Functions of Respiratory Data

Version 1.0.1

Date 2021-2-28

Author Xiaohua Douglas Zhang [aut, cph],
Teng Zhang [aut],
Xinzheng Dong [aut, cre]

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

Description Provides functions for the complete analysis of respiratory data. Consists of a set of functions that allow to preprocessing respiratory data, calculate both regular statistics and nonlinear statistics, conduct group comparison and visualize the results. Especially, Power Spectral Density ('PSD') (A. Eke (2000) <doi:10.1007/s004249900135>), 'MultiScale Entropy(MSE)' ('Madalena Costa(2002)' <doi:10.1103/PhysRevLett.89.068102>) and 'MultiFractal Detrended Fluctuation Analysis(MFDFA)' ('Jan W.Kantelhardt' (2002) <doi:10.1016/S0378-4371(02)01383-3>) were applied for the analysis of respiratory data.

License MIT + file LICENSE

Encoding UTF-8

LazyData true

Imports Rcpp (>= 1.0.2), signal, pracma

LinkingTo Rcpp

RoxygenNote 7.1.1

Depends R (>= 3.5.0)

NeedsCompilation yes

Repository CRAN

Date/Publication 2021-03-01 14:00:09 UTC

R topics documented:

`RespirAnalyzer-package`

Analysis Functions of Respiratory Data

Description

Provides functions for the complete analysis of respiratory data. Consists of a set of functions that allow to preprocessing respiratory data, calculate both regular statistics and nonlinear statistics, conduct group comparison and visualize the results. Especially, Power Spectral Density ('PSD') (A. Eke (2000) <doi:10.1007/s004249900135>), 'MultiScale Entropy(MSE)' ('Madalena Costa(2002)' <doi:10.1103/PhysRevLett.89.068102>) and 'MultiFractal Detrended Fluctuation Analysis(MFDFA)' ('Jan W.Kantelhardt' (2002) <doi:10.1016/S0378-4371(02)01383-3>) were applied for the analysis of respiratory data.

Details

The R package RespirAnalyzer contains functions for analyzing respiratory data.

Author(s)

Xiaohua Douglas Zhang [aut, cph], Teng Zhang [aut], Xinzheng Dong [aut, cre]

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

Examples

```
# load Data from TestData dataset
data("TestData")
Seriesplot.fn(Data[1:2000,1],Data[1:2000,2],points=FALSE,
              xlab="Time(s)",ylab="Respiratory")
Fs=50 ## sampling frequency is 50Hz
Peaks <- find.peaks(Data[,2],Fs,lowpass=TRUE,freq=1,MovingAv=FALSE,
                    W=FALSE,filter=TRUE,threshold=0.05)
points(Data[Peaks[2:13,1],1],Data[Peaks[2:13,1],2],col=2)
PP_interval <- diff(Peaks[,1])/Fs
```

```
Seriesplot.fn(1:length(PP_interval),PP_interval,points=FALSE,xlab="Count",
         ylab="Inter-breath Interval(s)")
#### Moving Average
W <- FS <- 50
Data[,3] <- MovingAverage(Data[,2],W)
Seriesplot.fn(Data[1:2000,1],Data[1:2000,2],points=FALSE,
              xlab="Time(s)",ylab="Respiratory")
lines(Data[1:2000,1],Data[1:2000,3],col=2)
#### Low pass filter
bf <- signal::butter(2, 2/Fs, type="low")
Data[,4] <- signal::filtfilt(bf,Data[,2])
Seriesplot.fn(Data[1:2000,1],Data[1:2000,2],points=FALSE,
              xlab="Time(s)",ylab="Respiratory")
lines(Data[1:2000,1],Data[1:2000,4],col=2)
#### entropy of  rawdata
scale_raw <- seq(1,90,2)
MSE <-  MSE(Data$V2[seq(1,100000,2)], tau=scale_raw, m=2, r=0.15, I=40000)
Seriesplot.fn(MSE$tau ,MSE$SampEn,points=TRUE,
              xlab="Scale",ylab="Sample entropy")
#### entropy of IBI
scale_PP <- 1:10
MSE <-  MSE(PP_interval, tau=scale_PP, m=2, r=0.15, I=40000)
Seriesplot.fn(MSE$tau ,MSE$SampEn,points=TRUE,
              xlab="Scale",ylab="Sample entropy")

#### PSD analysis
LowPSD(PP_interval, plot=TRUE,min=1/64, max=1/2)
#### MFDFA
exponents=seq(3, 9, by=1/4)
scale=2^exponents
q=-10:10
m=2
Result <- MFDFA(PP_interval, scale, m, q)
MFDFAplot.fn(Result,scale,q,model = TRUE)
#### fit.model
Coeff <- fit.model(Result$Hq,q)
Coeff
Para<- -log(Coeff)/log(2);Para[3]=Para[1]-Para[2]
names(Para)<-c("Hmax","Hmin","i÷H")
Para

#### Individualplot
data("HqData")
PP_Hq <- HqData
filenames <- row.names(PP_Hq)
q=-10:10
ClassNames <- c(substr(filenames[1:19], start = 1, stop = 3),
                substr(filenames[20:38], start = 1, stop = 5))
Class <- unique(ClassNames)
col_vec <- rep(NA, nrow(PP_Hq) )
pch_vec <- rep(16, nrow(PP_Hq) )
for( i in 1:length(Class) ) { col_vec[ ClassNames == Class[i] ] <- i }
Individualplot.fn(q,PP_Hq,Name=Class,col=col_vec,pch=pch_vec, xlab="q",ylab="Hurst exponent")
```

```
legend("topright", legend=paste0(Class, "(N=", table( ClassNames ), ")"),
      col=1:4, cex=1, lty=1, pch=16)

#### Groupplot
data("HqData")
PP_Hq <- HqData
filenames <- row.names(PP_Hq)
q <- -10:10
ClassNames <- c(substr(filenames[1:19], start = 1, stop = 3),
                substr(filenames[20:38], start = 1, stop = 5))
Class <- unique(ClassNames)
for (i in 1:length(q)){
  Data <- GroupComparison.fn(PP_Hq[,i],ClassNames)
  Result_mean_vec <- Data[,"Mean"]
  Result_sd_vec <- Data[,"SE"]
  if( i == 1 ) {
    Result_mean_mat <- Result_mean_vec
    Result_sd_mat <- Result_sd_vec
  } else {
    Result_mean_mat <- rbind(Result_mean_mat, Result_mean_vec)
    Result_sd_mat <- rbind(Result_sd_mat, Result_sd_vec)
 }
}
Groupplot.fn (q[1:10],Result_mean_mat[1:10,],Class,errorbar = Result_sd_mat[1:10,],
              xRange = NA, yRange = NA, col = NA, pch = rep(16,4), Position = "topright",
               cex.legend = 1, xlab="q",ylab="Hurst exponent",main = "")
Groupplot.fn (q[11:21],Result_mean_mat[11:21,],Class,errorbar = Result_sd_mat[11:21,],
              xRange = NA, yRange = NA, col = NA, pch = rep(16,4), Position = "topright",
               cex.legend = 1, xlab="q",ylab="Hurst exponent",main = "")
```

`Data` *An example of respiratory data*

Description

The respiratory data of a healthy people.

Details

This is an data to be included in my package

Source

Fantasia Database, PhysioNet

References

"https://doi.org/10.13026/C2RG61"

`find.peaks` *Function to find the peak-to-peak intervals of a respiratory signal.*

Description

function to find the peak-to-peak intervals of a respiratory signal.

Usage

```
find.peaks(
  y,
  Fs,
  lowpass = TRUE,
  freq = 1,
  MovingAv = FALSE,
  W = FALSE,
  filter = TRUE,
  threshold = 0.2
)
```

Arguments

`y`	a numeric vector, with respiratory data for a regularly spaced time series..
`Fs`	a positive value. sampling frequency of airflow signal.
`lowpass`	logical. Whether to use low-pass filtering to preprocess the airflow signal.
`freq`	an optional values. Cut-off frequency of low-pass filter. The default value is 1.
`MovingAv`	logical.Whether to use Moving Average to preprocess the airflow signal.
`W`	an optional values. the windows of Moving Average. The default value is equal to the sampling frequency Fs.
`filter`	logical.Whether to filter the points of peaks.
`threshold`	an optional value. A threshold is the minimum height difference between the wave crest and wave trough. The default value is 0.2.

Value

a dataframe for the information of peaks. "PeakIndex" is the position of the peaks and "PeakHeight" is the height of the peaks

Examples

```
data("TestData") # load Data from TestData dataset
Fs=50 ## sampling frequency is 50Hz
Peaks <- find.peaks(Data[,2],Fs,lowpass=TRUE,freq=1,MovingAv=FALSE,
                    W=FALSE,filter=TRUE,threshold=0.05)
Peaks
```

`fit.model` *Function to fit the MFDFA result with the extended binomial multifractal model.*

Description

function to fit the result of Multifractal detrended fluctuation analysis (MFDFA) with the extended binomial multifractal model. Return the results as a vector which contain the parameters of the model and the goodness of fit

Usage

```
fit.model(Hq, q)
```

Arguments

Hq	a nurmeric vector for the generalized Hurst exponent.
q	a vector of integers, q-order of the moment.

Value

a vector for fitting parameters."a" and "b" is the coefficients of the extended binomial multifractal model. "Goodness" is the goodness of fit

Examples

```
data("TestData") # load Data from TestData dataset
Fs=50 ## sampling frequency is 50Hz
Peaks=find.peaks(Data[,2],Fs)
PP_interval=diff(Peaks[,1])/Fs
exponents=seq(3, 8, by=1/4)
scale=2^exponents
q=-10:10
m=2
Result <- MFDFA(PP_interval, scale, m, q)
Coeff <- fit.model(Result$Hq,q)
Coeff
```

`GroupComparison.fn` *Function to calculate the statistics for each Group*

Description

function to calculate the statistics for each Group: Number of Samples, mean, standard deviation (SD), standard error (SE), median, confident interval, p-value of ANOVA

Usage

```
GroupComparison.fn(Data, GroupName, na.rm = TRUE, conf.level = 0.95)
```

Arguments

Data	vector for response values
GroupName	vector for group names
na.rm	whether to remove value for calculation
conf.level	confidence level

Value

a dataframe for the statistics for each Group. Number of Samples, mean

standard deviation (SD), median, upper and lower bounds of CI, p-value of ANOVA

Examples

```
data("HqData")
PP_Hq <- HqData
filenames <- row.names(PP_Hq)
q <- -10:10
ClassNames <- c(substr(filenames[1:19], start = 1, stop = 3),
                substr(filenames[20:38], start = 1, stop = 5))
Class <- unique(ClassNames)
Data <- GroupComparison.fn(PP_Hq[,1],ClassNames)
Data
```

Groupplot.fn *Function to plot the mean and error bar by group*

Description

function to plot the mean and error bar of sample entropy or the MFDFA results by group

Usage

```
Groupplot.fn(
  x,
  Average,
  GroupName,
  errorbar = NA,
  xRange = NA,
  yRange = NA,
  col = NA,
  pch = NA,
  Position = "topright",
```

```
  cex.legend = 0.75,
  xlab = "",
  ylab = "",
  main = ""
)
```

Arguments

x	a vector for x axis.
Average	Matrix for average in each group
GroupName	a vector of names for each group
errorbar	matrix for value of erorr bar
xRange	range for the x-axis
yRange	range for the y-axis
col	a vector for the colors to indicate groups
pch	a vector for points types to indicate groups
Position	position for the legend
cex.legend	cex for legend
xlab	a title for the x axis
ylab	a title for the y axis
main	main title for the plot

Value

No value returned

Examples

```
data("HqData")
PP_Hq <- HqData
filenames <- row.names(PP_Hq)
q <- -10:10
ClassNames <- c(substr(filenames[1:19], start = 1, stop = 3),
                substr(filenames[20:38], start = 1, stop = 5))
Class <- unique(ClassNames)
for (i in 1:length(q)){
  Data <- GroupComparison.fn(PP_Hq[,i],ClassNames)
  Result_mean_vec <- Data[,"Mean"]
  Result_sd_vec <- Data[,"SE"]
  if( i == 1 ) {
    Result_mean_mat <- Result_mean_vec
    Result_sd_mat <- Result_sd_vec
  } else {
    Result_mean_mat <- rbind(Result_mean_mat, Result_mean_vec)
    Result_sd_mat <- rbind(Result_sd_mat, Result_sd_vec)
 }
}
```

```
Groupplot.fn (q[1:10],Result_mean_mat[1:10,],Class,errorbar = Result_sd_mat[1:10,],
              xRange = NA, yRange = NA, col = NA, pch = rep(16,4), Position = "topright",
               cex.legend = 1, xlab="q",ylab="Hurst exponent",main = "")
Groupplot.fn (q[11:21],Result_mean_mat[11:21,],Class,errorbar = Result_sd_mat[11:21,],
              xRange = NA, yRange = NA, col = NA, pch = rep(16,4), Position = "topright",
               cex.legend = 1, xlab="q",ylab="Hurst exponent",main = "")
```

`HqData` *The Hurst exponent of respiratory data*

Description

The Hurst exponent extracted from the MFDFA result of respiratory data.

Details

This is an data to be included in my package

Source

Fantasia Database, PhysioNet

References

"https://doi.org/10.13026/C2RG61"

`Individualplot.fn` *Function to plot multiscale entropy or MFDFA results by individual.*

Description

function to plot multiscale entropy or MFDFA results by individual.

Usage

```
Individualplot.fn(
  x,
  y,
  Name = NA,
  xRange = NA,
  yRange = NA,
  col = NA,
  pch = NA,
  Position = "topright",
  cex.legend = 0.75,
```

```
  xlab = "",
  ylab = "",
  main = ""
)
```

Arguments

x	a vector for x-axis coordinate.
y	Matrix for response values.
Name	vector of names for each line.
xRange	range for the x-axis.
yRange	range for the y-axis.
col	vector for the colors to indicate groups.
pch	vector for points types to indicate groups.
Position	position for the legend.
cex.legend	cex for legend.
xlab	a title for x axis.
ylab	a title for y axis.
main	main title for the picture.

Value

No value returned

Examples

```
data("HqData")
PP_Hq <- HqData
filenames <- row.names(PP_Hq)
q=-10:10
ClassNames <- c(substr(filenames[1:19], start = 1, stop = 3),
                substr(filenames[20:38], start = 1, stop = 5))
Class <- unique(ClassNames)
col_vec <- rep(NA, nrow(PP_Hq) )
pch_vec <- rep(16, nrow(PP_Hq) )
for( i in 1:length(Class) ) { col_vec[ ClassNames == Class[i] ] <- i }
Individualplot.fn(q,PP_Hq,Name=Class,col=col_vec,pch=pch_vec, xlab="q",ylab="Hurst exponent")
legend("topright", legend=paste0(Class, "(N=", table( ClassNames ), ")"),
      col=1:4, cex=1, lty=1, pch=16)
```

LowPSD *Function to calculate the power spectral density (PSD)*

Description

function to calculate the power spectral density (PSD) of a time series. Methods derived from A. Eke (2000) "Physiological time series: distinguishing fractal noises from motions" <doi:10.1007/s004249900135>.

Usage

```
LowPSD(series, plot = TRUE, min = 1/8, max = 1/2)
```

Arguments

series	a numeric vector, with data for a regularly spaced time series.
plot	logical. whether to draw the plot of log power vs. log frequency
min, max	the optional values. Frequency range of power spectral density. The default value is 1/2 and 1/8 and cannot be set to a negative number

Value

a value of spectral exponent(beta) which the the slope of the of the fitting line on plot of log power vs. log frequency

Examples

```
data("TestData")
Fs <- 50
Peaks <- find.peaks(Data[,2],Fs,lowpass=TRUE,freq=1,MovingAv=FALSE,
                    W=FALSE,filter=TRUE,threshold=0.05)
PP_interval=diff(Peaks[,1])/Fs
LowPSD(series=PP_interval,plot=TRUE,min=1/64, max=1/2)
```

MFDFA *MultiFractal Detrended Fluctuation Analysis*

Description

Applies the MultiFractal Detrended Fluctuation Analysis (MFDFA) to time series.

Usage

```
MFDFA(tsx, scale, m, q)
```

Arguments

tsx	Univariate time series (must be a vector).
scale	Vector of scales.
m	An integer of the polynomial order for the detrending.
q	q-order of the moment.

Value

A list of the following elements:

- Hq q-order Hurst exponent.
- tau_q Mass exponent.
- hq Holder exponent.
- Dq singularity dimension.
- Fqi q-order fluctuation function.
- line linear fitting line of fluctuation function.

Examples

```
data("TestData") # load Data from TestData dataset
Fs <- 50
Peaks <- find.peaks(Data[,2],Fs,lowpass=TRUE,freq=1,MovingAv=FALSE,
                    W=FALSE,filter=TRUE,threshold=0.05)
head(Peaks)
PP_interval <- diff(Peaks$PeakIndex)/Fs
## Computing Multifractal
exponents=seq(3, 9, by=1/4)
scale=2^exponents
q=-10:10
m=2
Result <- MFDFA(PP_interval, scale, m, q)
Coeff <- fit.model(Result$Hq,q)
print(Coeff)
Para<- -log(Coeff)/log(2)
Para[3]=Para[1]-Para[2]
names(Para)<-c("Hmax","Hmin","DeltaH")
Para

PP_Hq <- Result$Hq
PP_hq <- Result$hq
PP_Dq <- Result$Dq
PP_Para <-Para
```

`MFDFAplot.fn` *Function to plot the results of MFDFA analysis*

Description

function to plot the results of MFDFA analysis: q-orde fluctuation function, Hurst exponent,mass exponent and multifractal spectrum. The fitting result of binomial multifratcal model can be also shown by the fitting line

Usage

```
MFDFAplot.fn(
  Result,
  scale,
  q,
  cex.lab = 1.6,
  cex.axis = 1.6,
  col.points = 1,
  col.line = 1,
  lty = 1,
  pch = 16,
  lwd = 2,
  model = TRUE,
  cex.legend = 1
)
```

Arguments

`Result`	a list of the MFDFA results.
`scale`	a vector of scales used to calculate the MFDFA results.
`q`	a vector, q-order of the moment used to calculate the MFDFA results.
`cex.lab`	the size of the tick label numbers/text with a numeric value of length 1.The default value is 1.6.
`cex.axis`	the size of the axis label text with a numeric value of length 1.The default value is 1.6.
`col.points`	color of the and point.
`col.line`	color of the line
`lty`	line types.
`pch`	points types.
`lwd`	line width.
`model`	whether to use the model to fit the results and draw a line of fit.
`cex.legend`	the size of the legend text with a numeric value of length 1.The default value is 1.

Value

No value returned

Examples

```
data("TestData")
Fs=50 ## sampling frequency is 50Hz
Peaks <- find.peaks(Data[,2],Fs,lowpass=TRUE,freq=1,MovingAv=FALSE,
                    W=FALSE,filter=TRUE,threshold=0.05)
PP_interval=diff(Peaks[,1])/Fs
exponents=seq(3, 9, by=1/4)
scale=2^exponents
q=-10:10
m=2
Result <- MFDFA(PP_interval, scale, m, q)
MFDFAplot.fn(Result,scale,q)
```

MovingAverage *Function to calculate Moving Average of a series*

Description

function to calculate Moving Average of a series

Usage

```
MovingAverage(y, W)
```

Arguments

y	a numeric vector, with respiratory data for a regularly spaced time series.
W	a Positive integer, the windows of Moving Average.

Value

A new numeric vector after calculate the moving average.

Examples

```
data("TestData")
W <- 50
y <- MovingAverage(Data[,2],W)
```

MSE *Function to compute the multiscale entropy(MSE)*

Description

function to perform a multiscale entropy (MSE) analysis of a regularly spaced time series. Return the results as an R data frame. Methods derived from Madalena Costa(2002) "Multiscale entropy analysis of complex physiologic time series" <doi:10.1103/PhysRevLett.89.068102>.

Usage

```
MSE(x, tau, m, r, I)
```

Arguments

x	a numeric vector, with data for a regularly spaced time series. NA's are not allowed (because the C program is not set up to handle them).
tau	a vector of scale factors to use for MSE. Scale factors are positive integers that specify bin size for the MSE algorithm: the number ofconsecutive observations in 'x' that form a bin and are averaged in the first step of the algorithm. Must be a sequence of equally-spaced integers starting at 1. The largest value must still leave a sufficient number of bins to estimate entropy.
m	a positive integers giving the window size for the entropy calculations in the second step of the algorithm, Typical values are 1, 2, or 3.
r	a positive value of coefficients for similarity thresholds, such as r=0.15, r*sd(y) must be in the same units as 'x'. Averages in two bins are defined to be similar if they differ by 'r*sd(y)' or less. NOTE: Currently only a single threshold is allowed per run; i.e.,'r' must be a scalar.
I	the maximal number of points to be used for calculating MSE

Value

A data frame with with one row for each combination of 'tau', 'm' and 'rSD'. Columns are "tau", "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(y)

Examples

```
data("TestData") # load Data from TestData dataset
oldoptions <- options(scipen=999)
Fs <- 50 # sampling frequency
scale_raw <- seq(1,90,by=2)
MSER <-  MSE(Data[1:10000,2], tau=scale_raw, m=2,
            r=0.15, I=40000)
print(MSER)
options(oldoptions)
```

`Seriesplot.fn` *Function to plot series data*

Description

function to plot series data. including Respiration data and Peak-to-Peak intervals series.

Usage

```
Seriesplot.fn(
  x,
  y,
  xRange = NA,
  yRange = NA,
  points = TRUE,
  pch = 1,
  col.point = 1,
  cex.point = 1,
  line = TRUE,
  lty = 1,
  col.line = 1,
  lwd = 1,
  xlab = "x",
  ylab = "y",
  main = ""
)
```

Arguments

`x`	a vector for the x-axis coordinate of a sequence.
`y`	a vector for the y-axis coordinates of a sequence.
`xRange`	range for the x-axis.
`yRange`	range for the y-axis.
`points`	whether to draw the points of the sequence. If points = TRUE, a sequence of points will be plotted. otherwise, will not plot the points.
`pch`	points types.
`col.point`	color code or name of the points.
`cex.point`	cex of points
`line`	whether to draw a line of the sequence. If line = TRUE, a line of sequence will be plotted. otherwise, will not plot the line.
`lty`	line types.
`col.line`	color code or name of the line.
`lwd`	line width.

xlab	a title for the x axis.
ylab	a title for the y axis.
main	main title for the picture.

Value

No value return

Examples

```
data("TestData")
oldpar <- par(mfrow=c(1,2))
Seriesplot.fn(Data[1:10000,1],Data[1:10000,2],points=FALSE,xlab="Time(s)",ylab="Respiration")
Fs=50 ## sampling frequency is 50Hz
Peaks <- find.peaks(Data[,2],Fs,lowpass=TRUE,freq=1,MovingAv=FALSE,
                    W=FALSE,filter=TRUE,threshold=0.05)
PP_interval=diff(Peaks[,1])/Fs
Seriesplot.fn(1:length(PP_interval),PP_interval,points=FALSE,xlab="Count",ylab="Interval(s)")
par(oldpar)
```

Index