Package 'psdr'

October 14, 2022

Title Use Time Series to Generate and Compare Power Spectral Density

Version 1.0.1

Description Functions that allow you to generate and compare power spectral density (PSD) plots given time series data. Fast Fourier Transform (FFT) is used to take a time series data, analyze the oscillations, and then output the frequencies of these oscillations in the time series in the form of a PSD plot.Thus given a time series, the dominant frequencies in the time series can be identified. Additional functions in this package allow the dominant frequencies of multiple groups of time series to be compared with each other. To see example usage with the main functions of this package, please visit this site: <https://yhhc2.github.io/psdr/articles/Introduction.html>. The mathematical operations used to generate the PSDs are described in these sites: <https://www.mathworks.com/help/matlab/ref/fft.html>. <https://www.mathworks.com/help/signal/ug/power-spectral-density-estimates-using-fft.html>.

License GPL (>= 3) | file LICENSE

Encoding UTF-8

RoxygenNote 7.1.1

Imports devtools (>= 2.4.1), ggplot2 (>= 3.3.2), qpdf (>= 1.1), stats (>= 4.0.2)

Suggests rmarkdown, knitr, testthat (>= 3.0.0)

VignetteBuilder knitr

Config/testthat/edition 3

NeedsCompilation no

Author Yong-Han Hank Cheng [aut, cre] (<https://orcid.org/0000-0001-7686-0697>)

Maintainer Yong-Han Hank Cheng <yhhc@uw.edu>

Repository CRAN

Date/Publication 2021-06-04 07:50:02 UTC

R topics documented:

`AutomatedCompositePlotting`

Automated plotting of time series, PSD, and log transformed PSD

Description

This function uses a lot of the functions in this package (psdr) to automate the plotting process for plotting composite curves and having multiple curves on the same plot.

Usage

```
AutomatedCompositePlotting(
  list.of.windows,
  name.of.col.containing.time.series,
  x_start = 0,
  x_end,
  x_increment,
  level1.column.name,
  level2.column.name,
  level.combinations,
  level.combinations.labels,
  plot.title,
  plot.xlab,
  plot.ylab,
  combination.index.for.envelope = NULL,
  TimeSeries.PSD.LogPSD = "TimeSeries",
```

```
  sampling_frequency = NULL,
  my.colors = c("blue", "red", "black", "green", "gold", "darkorchid1", "brown",
    "deeppink", "gray")
)
```

Arguments

list.of.windows	A list of windows (dataframes). All windows should have the same length, but because interpolation is used, the function still works if window length differs.
name.of.col.containing.time.series	A string that specifies the name of the column in the windows that correspond to the time series that should be used.
x_start	Numeric value specifying start of the new x-axis. Default is 0.
x_end	Numeric value specifying end of the new x-axis. For PSD, maximum value is the sampling_frequency divided by 2.
x_increment	Numeric value specifying increment of the new x-axis.
level1.column.name	A String that specifies the column name to use for the first level. This column should only contain one unique value within each window.
level2.column.name	A String that specifies the column name to use for the second level. This column should only contain one unique value within each window.
level.combinations	A List containing Lists. Each list that it contains has two vectors. The first vector specifying the values for level1 and the second vector specifying the values for level2. Each list element will correspond to a new line on the plot.
level.combinations.labels	A vector of strings that labels each combination. This is used for making the figure legend.
plot.title	String for title of plot.
plot.xlab	String for x-axis of plot.
plot.ylab	String for y-axis of plot.
combination.index.for.envelope	A numeric value that specifies which combination (index of level.combinations) should have a line with an error envelope. The default is no envelope.
TimeSeries.PSD.LogPSD	A String with 3 possible values to specify what type of plot to create from the time series: 1. "TimeSeries", 2. "PSD", 3. "LogPSD"
sampling_frequency	Numeric value used for specifying sampling frequency if PSD or LogPSD is made with this function. Default is NULL because default plot created is a time series plot.
my.colors	A vector of strings that specify the color for each line. 9 default values are used.

Details

Given a list of windows, you can specify which windows you want to average together to form a curve on the plot. You can specify multiple combos and therefore multiple curves can be plotted on the same plot with a legend to specify the combo used to create each curve. An error envelope can also be created for a single curve on the plot.

The function automatically generates a ggplot for easy plotting. However, the function also outputs dataframes for each combo. Each dataframe has 3 columns:

1. X value: For timeseries, this will be in the original units that separates each observation in the time series. For example, if there are 150 observations and each observation is 0.02 seconds apart, then if 150 observations are specified as the x_increment, then each observation are still 0.02 seconds. The time difference between the first and last observation needs to equal the time difference between the first and last observation in the original time series. For PSD and LogPSd, the units will be in Hz (frequency). The frequency range depends on the sampling frequency. Smallest frequency is 0 and largest frequency is sampling_frequency/2.
2. Y value: For time series, this will be in the original units of the time series. For PSD, the units will be (original units)^2/Hz, for LogPSD, the units will be log((original units)^2/Hz)).
3. Standard deviation of Y value. This can be used to plot error bars or error envelopes to see the spread of the windows used to make the composite.

Three different plots can be created:

1. Time series plot. This simply takes the time series in the windows, averages them for each combo, and then plots the composite curve for each combo.
2. PSD plot. This takes the time series in the windows and given the sampling frequency, it calculates the PSD. It averages the PSD for the windows in each combo, and then plots the composite curve for each combo.
3. Log transformed PSD plot. Same as PSD plot except at the end, the composite PSD curves are log transformed.

Value

A List with three objects:

1. A List of dataframes containing values for each line on the plot. The order of the dataframes correspond to the order of the combinations in level.combinations.
2. A ggplot object that can be plotted right away.
3. If plot selected is a PSD, then a List is outputted from SingleBinPSDIntegrationOrDominant-FreqComparison() to compare dominant frequencies between combinations.

Examples

```
#I want to create a plot that shows two curves:
#1. Composite of time series signals 1, 2, and 3.
#2. Composite of time series signals 3 and 4.

#Create a vector of time that represent times where data are sampled.
```

```
Fs = 100; #sampling frequency in Hz
T = 1/Fs; #sampling period
L = 1000; #length of time vector
t = (0:(L-1))*T; #time vector

#First signal
#1. 1 Hz with amplitude of 2
S1 <- 2*sin(2*pi*1*t)
level1.vals <- rep("a", length(S1))
level2.vals <- rep("1", length(S1))
S1.data.frame <- as.data.frame(cbind(t, S1, level1.vals, level2.vals))
colnames(S1.data.frame) <- c("Time", "Signal", "level1.ID", "level2.ID")
S1.data.frame[,"Signal"] <- as.numeric(S1.data.frame[,"Signal"])

#Second signal
#1. 1 Hz with amplitude of -4
#2. 2 Hz with amplitude of -2
S2 <- (-4)*sin(2*pi*1*t) - 2*sin(2*pi*2*t);
level1.vals <- rep("a", length(S2))
level2.vals <- rep("2", length(S2))
S2.data.frame <- as.data.frame(cbind(t, S2, level1.vals, level2.vals))
colnames(S2.data.frame) <- c("Time", "Signal", "level1.ID", "level2.ID")
S2.data.frame[,"Signal"] <- as.numeric(S2.data.frame[,"Signal"])

#Third signal
#1. 1 Hz with amplitude of 2
#2. 2 Hz with amplitude of 2
S3 <- 2*sin(2*pi*1*t) + 2*sin(2*pi*2*t);
level1.vals <- rep("a", length(S3))
level2.vals <- rep("3", length(S3))
S3.data.frame <- as.data.frame(cbind(t, S3, level1.vals, level2.vals))
colnames(S3.data.frame) <- c("Time", "Signal", "level1.ID", "level2.ID")
S3.data.frame[,"Signal"] <- as.numeric(S3.data.frame[,"Signal"])

#Fourth signal
#1. 1 Hz with amplitude of -2
S4 <- -2*sin(2*pi*1*t)
level1.vals <- rep("b", length(S4))
level2.vals <- rep("3", length(S4))
S4.data.frame <- as.data.frame(cbind(t, S4, level1.vals, level2.vals))
colnames(S4.data.frame) <- c("Time", "Signal", "level1.ID", "level2.ID")
S4.data.frame[,"Signal"] <- as.numeric(S4.data.frame[,"Signal"])

windows <- list(S1.data.frame, S2.data.frame, S3.data.frame, S4.data.frame)

#Gets the composite of the first, second, and third signal. Should result in a flat signal.
FirstComboToUse <- list( c("a"), c(1, 2, 3) )

#Gets the composite of the third and fourth signal
SecondComboToUse <- list( c("a", "b"), c(3) )


#Timeseries----------------------------------------------------------------
```

```
timeseries.results <- AutomatedCompositePlotting(list.of.windows = windows,
                            name.of.col.containing.time.series = "Signal",
                            x_start = 0,
                            x_end = 999,
                            x_increment = 1,
                            level1.column.name = "level1.ID",
                            level2.column.name = "level2.ID",
                            level.combinations = list(FirstComboToUse, SecondComboToUse),
                          level.combinations.labels = c("Signal 1 + 2 + 3", "Signal 3 + 4"),
                            plot.title = "Example",
                            plot.xlab = "Time",
                            plot.ylab = "Original units",
                            combination.index.for.envelope = NULL,
                            TimeSeries.PSD.LogPSD = "TimeSeries",
                            sampling_frequency = NULL)

ggplot.obj.timeseries <- timeseries.results[[2]]

#Plot. Will see the 1+2+3 curve as a flat line. The 3+4 curve will only have 2 Hz.
##dev.new()
ggplot.obj.timeseries

#PSD--------------------------------------------------------------------------

#Note that the PSDs are not generated directly from the "Signal 1 + 2 + 3" and
#the "Signal 3 + 4" time series. Instead, PSDs are generated individually
#for signals 1, 2, 3, and 4, and then then are summed together.

PSD.results <- AutomatedCompositePlotting(list.of.windows = windows,
                            name.of.col.containing.time.series = "Signal",
                            x_start = 0,
                            x_end = 50,
                            x_increment = 0.01,
                            level1.column.name = "level1.ID",
                            level2.column.name = "level2.ID",
                            level.combinations = list(FirstComboToUse, SecondComboToUse),
                          level.combinations.labels = c("Signal 1 + 2 + 3", "Signal 3 + 4"),
                            plot.title = "Example",
                            plot.xlab = "Hz",
                            plot.ylab = "(Original units)^2/Hz",
                            combination.index.for.envelope = 2,
                            TimeSeries.PSD.LogPSD = "PSD",
                            sampling_frequency = 100)

ggplot.obj.PSD <- PSD.results[[2]]

#Plot. For both plots, two peaks will be present, 1 Hz and 2 Hz. 1 Hz should be
#stronger in both cases because more signals have this frequency (even if amp is negative).
#Error envelope is specified for the second (red) curve. Envelope should only
#be present for 2 Hz signal.
#dev.new()
ggplot.obj.PSD
```

```
#PSD Zoomed in-----------------------------------------------------------------

PSD.results <- AutomatedCompositePlotting(list.of.windows = windows,
                            name.of.col.containing.time.series = "Signal",
                            x_start = 0,
                            x_end = 5,
                            x_increment = 0.01,
                            level1.column.name = "level1.ID",
                            level2.column.name = "level2.ID",
                            level.combinations = list(FirstComboToUse, SecondComboToUse),
                          level.combinations.labels = c("Signal 1 + 2 + 3", "Signal 3 + 4"),
                            plot.title = "Example",
                            plot.xlab = "Hz",
                            plot.ylab = "(Original units)^2/Hz",
                            combination.index.for.envelope = 2,
                            TimeSeries.PSD.LogPSD = "PSD",
                            sampling_frequency = 100)

ggplot.obj.PSD <- PSD.results[[2]]

#Plot. For both plots, two peaks will be present, 1 Hz and 2 Hz. 1 Hz should be
#stronger in both cases because more signals have this frequency (even if amp is negative).
#Error envelope is specified for the second (red) curve. Envelope should only
#be present for 1 Hz signal.
#dev.new()
ggplot.obj.PSD

#LogPSD-------------------------------------------------------------------------

LogPSD.results <- AutomatedCompositePlotting(list.of.windows = windows,
                            name.of.col.containing.time.series = "Signal",
                            x_start = 0,
                            x_end = 50,
                            x_increment = 0.01,
                            level1.column.name = "level1.ID",
                            level2.column.name = "level2.ID",
                            level.combinations = list(FirstComboToUse, SecondComboToUse),
                          level.combinations.labels = c("Signal 1 + 2 + 3", "Signal 3 + 4"),
                            plot.title = "Example",
                            plot.xlab = "Hz",
                            plot.ylab = "log((Original units)^2/Hz)",
                            combination.index.for.envelope = NULL,
                            TimeSeries.PSD.LogPSD = "LogPSD",
                            sampling_frequency = 100)

ggplot.obj.LogPSD <- LogPSD.results[[2]]

#Plot. For both plots, two peaks will be present, 1 Hz and 2 Hz. 1 Hz should
#be stronger in both cases because more signals have this frequency (even if amp is negative).
#Error envelope is specified for the second (red) curve. Envelope should only
#be present for 2 Hz signal.
#dev.new()
```

ggplot.obj.LogPSD

CountWindows *Create a contingency table to display how many windows (dataframes) fall into particular categories*

Description

Given a List of homogeneous windows (dataframes where the two selected columns in each dataframe only have one unique value), make a contingency table to show how many windows fall into each of the categories in level1 and level2.

Usage

```
CountWindows(
  list.of.windows,
  level1.column.name,
  level2.column.name,
  level1.categories,
  level2.categories
)
```

Arguments

`list.of.windows`
: A list of windows (dataframes) and each window can only have a single unique value for the two specified columns.

`level1.column.name`
: A String that specifies the column name to use for the first level of the contingency table. This column should only contain one unique value within each window.

`level2.column.name`
: A String that specifies the column name to use for the second level of the contingency table. This column should only contain one unique value within each window.

`level1.categories`
: A vector that specifies the possible labels in the level1 column. This will be used as the rows of the outputted matrix.

`level2.categories`
: A vector that specifies the possible labels in the level2 column. This will be used as the columns of the outputted matrix.

Details

A List of homogeneous windows is inputted. For each window in the list, the columns specified by level1.column.name and level2.column.name can only have one value (definition of homogeneous window). The value of the level1.column and level2.column for each window is evaluated and the number of windows in each category is captured and outputted as a 2D matrix with level1 as the row labels and level2 as the column labels.

Value

A matrix which is formatted as a contingency table.

Examples

```
#Example using a dataframe with 5 homogeneous windows.

#Windows are homogeneous if looking at col.two and col.three values.
window.name.column <- c(10, 10, 10, 20, 20, 20, 30, 30, 30, 30, 40, 40, 50, 50, 50, 50)
col.two <- c("a", "a", "a", "a", "a", "a", "a", "a", "a", "a", "a", "b", "b", "a", "a", "a", "a")
col.three <- c(1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 1, 1, 3, 3, 3, 3)

multi.window.data <- data.frame(window.name.column, col.two, col.three)

list.of.homogeneous.windows <- GetHomogeneousWindows(multi.window.data,
"window.name.column", c("col.two", "col.three"))

matrix <- CountWindows(list.of.homogeneous.windows, "col.two", "col.three",
c("a", "b"), c("1", "2", "3"))

matrix
```

`FindHomogeneousWindows` *Assess if window (dataframe) share certain features across all observations*

Description

For a given window (dataframe of observations where rows are observations), evaluate whether all observations in the window share the same values for specified columns.

Usage

```
FindHomogeneousWindows(inputted.data, names.of.columns.to.look.at)
```

Arguments

inputted.data	A dataframe.
names.of.columns.to.look.at	A vector of strings with each string being the name of a column in the datafarame to look at.

Details

Given a dataframe, look at the values in each of the specified column and sees if there is only one level in the specified column. If there is only one level, then this means rows share the same value for that column. Do this for all specified columns and return true if each column only contains one value. If all observations share the same value for the specified columns, then the window is considered a homogeneous window.

Value

Boolean (true/false) indicating if window is homogeneous.

Examples

```
col.one <- c(1, 2, 3, 4, 5)
col.two <- c("a", "a", "a", "a", "a")
col.three <- c(1, 1, 1, 1, 1)

single.window.data <- data.frame(col.one, col.two, col.three)

#Example of inhomogeneous window if looking at col.one and col.two because
#col.one does not only have a single unique value.
result <- FindHomogeneousWindows(single.window.data , c("col.one", "col.two"))

result

#Example of homogeneous window if looking at col.two and col.three because
#col.two and col.three both only have a single unique value.
result <- FindHomogeneousWindows(single.window.data , c("col.two", "col.three"))

result
```

GenerateExampleData *Produce example data set for demonstrating package functions*

Description

Produce example data set for demonstrating package functions

Usage

```
GenerateExampleData()
```

Value

A data frame

`GetHomogeneousWindows` *Get windows (dataframes) that have the same specified column values for all observations*

Description

For a given dataframe where the windows are specified by a column in the dataframe, evaluate whether all observations in each window share the same values for specified columns.

Usage

```
GetHomogeneousWindows(
  inputted.data,
  window.ID.col.name,
  observation.vals.to.compare
)
```

Arguments

`inputted.data`	A dataframe that needs a column that labels which window each observation belongs to.
`window.ID.col.name`	A string that specifies the column name of the column that provides the window name.
`observation.vals.to.compare`	A vector of strings with each string being the name of a column in the datafarame to look at.

Details

Function takes a single dataframe with each row as observations. This dataframe needs a column that specifies which window each observation belongs to. For each window, the observations within the window is evaluated to see if all observations share the same values for specified columns of the dataframe. Windows where the specified columns have the same values across all observations are labeled as "homogeneous" windows and are captured and outputted in a list, where each element is a window (dataframe). This function uses the FindHomogeneousWindows() function to determine whether each window is homogeneous. As the code executes, it outputs number indicating how many homogeneous windows have been found so far in the inputted.data.

Value

List where each object is a homogeneous window (dataframe) that has observations sharing the same values for the observation.vals.to.compare column(s).

Examples

```
#Example using a dataframe with 3 windows.

#Windows 20 and 30 are homogeneous if looking at col.two and col.three values.
window.name.column <- c(10, 10, 10, 20, 20, 20, 30, 30, 30, 30)
col.two <- c("a", "a", "a", "a", "a", "a", "a", "a", "a", "a")
col.three <- c(1, 1, 0, 1, 1, 1, 2, 2, 2, 2)

multi.window.data <- data.frame(window.name.column, col.two, col.three)

result <- GetHomogeneousWindows(multi.window.data, "window.name.column", c("col.two", "col.three"))

#As expected, it looks like two windows are homogeneous.
str(result)

#Output the two windows that are homogeneous:

result[[1]]

result[[2]]
```

GetSubsetOfWindows *Select only windows (dataframes) where a specified column matches a specified value*

Description

Looks at all the windows (dataframes) in a list of homogeneous windows. And only selects the windows where the column value contains a specified value.

Usage

```
GetSubsetOfWindows(
  list.of.windows,
  name.of.column.to.look.at.in.window,
  value.to.match.to
)
```

Arguments

`list.of.windows`
: A list of windows (dataframes) and each window can only have a single unique value for the name.of.column.to.look.at.in.window column.

`name.of.column.to.look.at.in.window`
: String that specifies which column to look at.

`value.to.match.to`
: String that specifies what value to look for in name.of.column.to.look.at.in.window.

Details

Takes a List of windows (dataframes) where each window is a homogeneous window, which means in each window, there is only one unique value in the specified column. This function looks through the homogeneous windows in the List, selects the windows that have a specified column value in the specified column, then puts these windows into a new List and outputs the new List of windows.

Value

List containing only selected windows (windows that have value.to.match.to value in the name.of.column.to.look.at.in.windo column).

Examples

```
#Example using a dataframe with 3 windows.

#Windows 20 and 30 are homogeneous if looking at col.two and col.three values.
window.name.column <- c(10, 10, 10, 20, 20, 20, 30, 30, 30, 30)
col.two <- c("a", "a", "a", "a", "a", "a", "a", "a", "a", "a")
col.three <- c(1, 1, 0, 1, 1, 1, 2, 2, 2, 2)

multi.window.data <- data.frame(window.name.column, col.two, col.three)

list.of.homogeneous.windows <- GetHomogeneousWindows(multi.window.data,
"window.name.column", c("col.two", "col.three"))

#Get a subset of windows where col.three has a value of 2
subset.list.of.homogeneous.windows <- GetSubsetOfWindows(list.of.homogeneous.windows,
"col.three", "2")

str(subset.list.of.homogeneous.windows)

subset.list.of.homogeneous.windows[[1]]
```

`GetSubsetOfWindowsTwoLevels` *Select only windows (dataframes) where a two specified columns must match specified values*

Description

Looks at all the windows (dataframes) in a list of homogeneous windows. And only selects the windows where the column values for two columns matches the specified values.

Usage

```
GetSubsetOfWindowsTwoLevels(
  list.of.windows,
  level1.column.name,
  level2.column.name,
  level1.categories,
  level2.categories
)
```

Arguments

`list.of.windows`
: A list of windows (dataframes) and each window can only have a single unique value for the two specified columns.

`level1.column.name`
: A String that specifies the column name to use for the first level. This column should only contain one unique value within each window.

`level2.column.name`
: A String that specifies the column name to use for the second level. This column should only contain one unique value within each window.

`level1.categories`
: A vector that specifies the possible labels in the level1 column.

`level2.categories`
: A vector that specifies the possible labels in the level2 column.

Details

Takes a List of windows (dataframes) where each window is a homogeneous window, which means in each window, there is only one unique value in the specified column. This function looks through the homogeneous windows in the List, selects the windows that have specified column value(s) in the first specified column, then from the windows selected based on the first column, windows are further selected to have specified column value(s) in the second specified column. Puts these windows into a new List and outputs the new List of windows. Uses the GetSubsetOfWindows() function.

Value

List containing only selected windows

Examples

```
#Example using a dataframe with 5 homogeneous windows.

#Windows are homogeneous if looking at col.two and col.three values.
window.name.column <- c(10, 10, 10, 20, 20, 20, 30, 30, 30, 30, 40, 40, 50, 50, 50, 50)
col.two <- c("a", "a", "a", "a", "a", "a", "a", "a", "a", "a", "a", "b", "b", "a", "a", "a", "a")
col.three <- c(1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 1, 1, 3, 3, 3, 3)

multi.window.data <- data.frame(window.name.column, col.two, col.three)

list.of.homogeneous.windows <- GetHomogeneousWindows(multi.window.data,
"window.name.column", c("col.two", "col.three"))

result <- GetSubsetOfWindowsTwoLevels(list.of.homogeneous.windows, "col.two", "col.three",
c("a"), c("1", "2"))

#Should contain windows 10, 20, 30 because col.two is "a" and col.three can be "1" or "2"
result
```

IdentifyMaxOnXY *Given a xy plot. Find the maximum value on the plot*

Description

To generate a curve of points, interpolation is used and the range and increment can be specified. Will output a message if multiple maxima are detected.

Usage

```
IdentifyMaxOnXY(x_vector, y_vector, x_start = 0, x_end, x_increment)
```

Arguments

x_vector	A numerical vector with x coordinates.
y_vector	A numerical vector with y coordinates.
x_start	Numeric value specifying start of x value to look at.
x_end	Numeric value specifying end of x value to look at.
x_increment	Numeric value specifying the increment of the x-values to use.

Value

A vector with two elements, The first element is the x value where the max y value is found. The second element is the max y value.

Examples

```
#I want to create a plot that shows two curves:
#1. Composite of time series signals 1, 2, and 3.
#2. Composite of time series signals 3 and 4.

#Create a vector of time that represent times where data are sampled.
Fs = 100; #sampling frequency in Hz
T = 1/Fs; #sampling period
L = 1000; #length of time vector
t = (0:(L-1))*T; #time vector

#First signal
#1. 1 Hz with amplitude of 2
S1 <- 2*sin(2*pi*1*t)
level1.vals <- rep("a", length(S1))
level2.vals <- rep("1", length(S1))
S1.data.frame <- as.data.frame(cbind(t, S1, level1.vals, level2.vals))
colnames(S1.data.frame) <- c("Time", "Signal", "level1.ID", "level2.ID")
S1.data.frame[,"Signal"] <- as.numeric(S1.data.frame[,"Signal"])

#Second signal
#1. 1 Hz with amplitude of -4
#2. 2 Hz with amplitude of -2
S2 <- (-4)*sin(2*pi*1*t) - 2*sin(2*pi*2*t);
level1.vals <- rep("a", length(S2))
level2.vals <- rep("2", length(S2))
S2.data.frame <- as.data.frame(cbind(t, S2, level1.vals, level2.vals))
colnames(S2.data.frame) <- c("Time", "Signal", "level1.ID", "level2.ID")
S2.data.frame[,"Signal"] <- as.numeric(S2.data.frame[,"Signal"])

#Third signal
#1. 1 Hz with amplitude of 2
#2. 2 Hz with amplitude of 2
S3 <- 2*sin(2*pi*1*t) + 2*sin(2*pi*2*t);
level1.vals <- rep("a", length(S3))
level2.vals <- rep("3", length(S3))
S3.data.frame <- as.data.frame(cbind(t, S3, level1.vals, level2.vals))
colnames(S3.data.frame) <- c("Time", "Signal", "level1.ID", "level2.ID")
S3.data.frame[,"Signal"] <- as.numeric(S3.data.frame[,"Signal"])

#Fourth signal
#1. 1 Hz with amplitude of -2
S4 <- -2*sin(2*pi*1*t)
level1.vals <- rep("b", length(S4))
level2.vals <- rep("3", length(S4))
S4.data.frame <- as.data.frame(cbind(t, S4, level1.vals, level2.vals))
colnames(S4.data.frame) <- c("Time", "Signal", "level1.ID", "level2.ID")
S4.data.frame[,"Signal"] <- as.numeric(S4.data.frame[,"Signal"])

#Extra representation of S2 dataframe to show an example that has enough samples
#to have significance for Kruskal-Wallis test
```

```
windows <- list(S1.data.frame, S2.data.frame, S2.data.frame, S2.data.frame, S2.data.frame,
S2.data.frame, S2.data.frame, S2.data.frame, S2.data.frame, S2.data.frame, S3.data.frame,
S4.data.frame)

#Gets the composite of the first, second, and third signal. Should result in a flat signal.
FirstComboToUse <- list( c("a"), c(1, 2, 3) )

#Gets the composite of the third and fourth signal
SecondComboToUse <- list( c("a", "b"), c(3) )


#PSD--------------------------------------------------------------------------

PSD.results <- AutomatedCompositePlotting(list.of.windows = windows,
                            name.of.col.containing.time.series = "Signal",
                            x_start = 0,
                            x_end = 10,
                            x_increment = 0.01,
                            level1.column.name = "level1.ID",
                            level2.column.name = "level2.ID",
                            level.combinations = list(FirstComboToUse, SecondComboToUse),
                          level.combinations.labels = c("Signal 1 + 2 + 3", "Signal 3 + 4"),
                            plot.title = "Example",
                            plot.xlab = "Hz",
                            plot.ylab = "(Original units)^2/Hz",
                            combination.index.for.envelope = 2,
                            TimeSeries.PSD.LogPSD = "PSD",
                            sampling_frequency = 100)

ggplot.obj.PSD <- PSD.results[[2]]

dataframes.plotted <- PSD.results[[1]]

first.curve <- dataframes.plotted[[1]]

second.curve <- dataframes.plotted[[2]]

first.curve.max <- IdentifyMaxOnXY(first.curve$xvals, first.curve$yvals, 0, 10, 0.01)
first.curve.max.limited <- IdentifyMaxOnXY(first.curve$xvals, first.curve$yvals, 1.25, 2.5, 0.01)

second.curve.max <- IdentifyMaxOnXY(second.curve$xvals, second.curve$yvals, 0, 10, 0.01)
```

MakeCompositePSDForAllWindows

Make PSD for each window (dataframe) in a list and then find the average of all the PSDs

---

Description

Given multiple windows of time series data, if the sampling frequency for all time series is the same, then the PSD for each window can be calculated, and then averaged to create a composite PSD.

Usage

```
MakeCompositePSDForAllWindows(
  list.of.windows,
  name.of.col.containing.time.series,
  sampling_frequency,
  x_start = 0,
  x_end,
  x_increment
)
```

Arguments

`list.of.windows`	A list of windows (dataframes).
`name.of.col.containing.time.series`	A string that specifies the name of the column in the windows that correspond to the time series that should be used for making PSD.
`sampling_frequency`	Numeric value specifying sampling frequency in hertz. If data is sampled once every second, then sampling frequency is 1 Hz. If data is sampled once every 2 seconds, then sampling frequency is 0.5 Hz.
`x_start`	Numeric value specifying start of the new x-axis for the averaged PSD. Default is 0 Hz.
`x_end`	Numeric value specifying end of the new x-axis for the averaged PSD. Maximum value is the sampling_frequency divided by 2.
`x_increment`	Numeric value specifying increment of the new x-axis for the averaged PSD.

Details

Using fft(), the PSD of a time series dataset can be calculated.This is done for multiple windows of time using the MakePowerSpectralDensity() function for each window. When the code executes, a counter is displayed to indicate how many windows have been analyzed.

Value

A List with two objects:

1. Vector of frequencies in Hz. The maximum frequency should be half the sampling frequency. Called Nyquist Frequency.
2. Vector of averaged PSD values corresponding with each frequency. Units should be in the original units of the data vector squared and divided by frequency.
3. Vector of standard deviation of PSD values corresponding with each frequency. This can be used to generate error envelopes or error bars to show the variation between windows.

The vector of frequencies can be used as the x-axis values of a single sided spectrum amplitude plot. The vector of PSD values can be used as the y-axis values of the PSD plot.

Examples

```
#Create a vector of time that represent times where data are sampled.
Fs = 100; #sampling frequency in Hz
T = 1/Fs; #sampling period
L = 1000; #length of time vector
t = (0:(L-1))*T; #time vector

#First signal
#1. 10 Hz with amplitude of 4
#2. 25 Hz with amplitude of 4
S1 <- 1*sin(2*pi*10*t) + 2*sin(2*pi*25*t);
S1 <- S1 + rnorm(length(t)) #Add some noise
S1.data.frame <- as.data.frame(cbind(t, S1))
colnames(S1.data.frame) <- c("Time", "Signal")

#Second signal
#1. 5 Hz with amplitude of 2
#2. 8 Hz with amplitude of 2
S2 <- 2*sin(2*pi*5*t) + 2*sin(2*pi*8*t);
S2 <- S2 + rnorm(length(t)) #Add some noise
S2.data.frame <- as.data.frame(cbind(t, S2))
colnames(S2.data.frame) <- c("Time", "Signal")

#Third signal
#1. 5 Hz with amplitude of 2
#2. 8 Hz with amplitude of 2
S3 <- 2*sin(2*pi*5*t) + 2*sin(2*pi*8*t);
S3 <- S3 + rnorm(length(t)) #Add some noise
S3.data.frame <- as.data.frame(cbind(t, S3))
colnames(S3.data.frame) <- c("Time", "Signal")

#Add all signals to a List
list.of.windows <- list(S1.data.frame, S2.data.frame, S3.data.frame)

results <- MakeCompositePSDForAllWindows(list.of.windows, "Signal", Fs, 0, 30, 0.1)

frequencies <- results[[1]]

averaged.PSD <- results[[2]]

stddev.PSD <- results[[3]]

#dev.new()
plot(frequencies, averaged.PSD, type = "l")

#dev.new()
plot(frequencies, averaged.PSD, type = "l")
#Add error bars
arrows(frequencies, averaged.PSD, frequencies, averaged.PSD + stddev.PSD, length=0.05, angle=90)
```

```
arrows(frequencies, averaged.PSD, frequencies, averaged.PSD - stddev.PSD, length=0.05, angle=90)
```

`MakeCompositeXYPlotForAllWindows` *Find averaged xy plots*

Description

If there are multiple 2D plots where the range of the x values are the same, then this function can allow you to average the y-values for all of these plots. The increment of the x-values can be different because this function uses interpolation to ensure each window has the same x-axis when the averaging step occurs.

Usage

```
MakeCompositeXYPlotForAllWindows(
  list.of.windows,
  name.of.col.containing.time.series,
  x_start = 0,
  x_end,
  x_increment
)
```

Arguments

`list.of.windows`	A list of windows (dataframes). Each window should have the same range of values in the x-axis in order for averaging to work.
`name.of.col.containing.time.series`	A string that specifies the name of the column in the windows that correspond to the time series that should be used for making averaging.
`x_start`	Numeric value specifying start of the new x-axis for the averaged PSD. Default is 0, so the first observation in the time series corresponds with x = 0.
`x_end`	Numeric value specifying end of the new x-axis for the averaged PSD. Maximum value is the sampling_frequency divided by 2.
`x_increment`	Numeric value specifying increment of the new x-axis for the averaged PSD.

Value

1. Vector of x values for plotting. The units will be number of observations. So if the time series has 100 observations and x_increment used is 1, then each tick mark on the x-axis corresponds to one observation unit.
2. Vector of averaged y values after looking at all windows.
3. Vector of standard deviation of y values for each x value.

Examples

```
#Create a vector of time that represent times where data are sampled.
Fs = 100; #sampling frequency in Hz
T = 1/Fs; #sampling period
L = 1000; #length of time vector
t = (0:(L-1))*T; #time vector

#First signal
#1. 1 Hz with amplitude of 4
S1 <- 4*sin(2*pi*1*t)
S1.data.frame <- as.data.frame(cbind(t, S1))
colnames(S1.data.frame) <- c("Time", "Signal")

#Second signal
#1. 1 Hz with amplitude of -2
#2. 2 Hz with amplitude of -2
S2 <- (-2)*sin(2*pi*1*t) - 2*sin(2*pi*2*t);
S2.data.frame <- as.data.frame(cbind(t, S2))
colnames(S2.data.frame) <- c("Time", "Signal")

#Third signal
#1. 1 Hz with amplitude of 2
#2. 2 Hz with amplitude of 2
S3 <- 2*sin(2*pi*1*t) + 2*sin(2*pi*2*t);
S3.data.frame <- as.data.frame(cbind(t, S3))
colnames(S3.data.frame) <- c("Time", "Signal")

#Add all signals to a List
list.of.windows <- list(S1.data.frame, S2.data.frame, S3.data.frame)

results <- MakeCompositeXYPlotForAllWindows(list.of.windows, "Signal", 0, 999, 1)

x.values <- results[[1]]

y.values <- results[[2]]

stddev.y.values <- results[[3]]

#plot each xy plot individually
#dev.new()
plot(t, S1, ylim = c(-5, 5), type = "l")
lines(t, S2, col="blue")
lines(t, S3, col="green")


#plot the averaged plot
#The only curve remaining should be the 1Hz with amplitude of 4/3.
#dev.new()
plot(x.values, y.values, type = "l")

#plot averaged plot with error bars
#dev.new()
```

```
plot(x.values, y.values, type = "l")
#Add error bars
arrows(x.values, y.values, x.values, y.values + stddev.y.values, length=0.05, angle=90)
arrows(x.values, y.values, x.values, y.values - stddev.y.values, length=0.05, angle=90)
```

`MakeOneSidedAmplitudeSpectrum` *Create a one sided amplitude spectrum using time series data*

Description

An explanation for some of the math can be found here: https://www.mathworks.com/help/matlab/ref/fft.html

Usage

```
MakeOneSidedAmplitudeSpectrum(sampling_frequency, data_vector)
```

Arguments

`sampling_frequency`	Numeric value specifying sampling frequency in hertz. If data is sampled once every second, then sampling frequency is 1 Hz. If data is sampled once every 2 seconds, then sampling frequency is 0.5 Hz.
`data_vector`	Vector of numeric values. Time series vector of data.

Value

A List with two objects:

1. Vector of frequencies in Hz. The maximum frequency should be half the sampling frequency. Called Nyquist Frequency.
2. Vector amplitudes corresponding with each frequency. Units should be in the original units of the data vector.

The vector of frequencies can be used as the x-axis values of a single sided spectrum amplitude plot. The vector of amplitudes can be used as the y-axis values of a single sided spectrum.

Examples

```
#Create a vector of time that represent times where data are sampled.
Fs = 100; #sampling frequency in Hz
T = 1/Fs; #sampling period
L = 1000; #length of time vector
t = (0:(L-1))*T; #time vector

#Form a signal (time series) that contains two frequencies:
```

```
#1. 10 Hz with amplitude of 1
#2. 25 Hz with amplitude of 2
S <- 1*sin(2*pi*10*t) + 2*sin(2*pi*25*t);

results <- MakeOneSidedAmplitudeSpectrum(Fs, S)

frequencies <- results[[1]]

amplitudes <- results[[2]]

#dev.new()
plot(frequencies, amplitudes, type = "l")
```

`MakePowerSpectralDensity` *Create a power spectral density (PSD) plot using time series data*

Description

Dividing the results of fft() by the frequency bin width, the PSD of a time series data set can be calculated.

Usage

```
MakePowerSpectralDensity(sampling_frequency, data_vector)
```

Arguments

`sampling_frequency`	Numeric value specifying sampling frequency in hertz. If data is sampled once every second, then sampling frequency is 1 Hz. If data is sampled once every 2 seconds, then sampling frequency is 0.5 Hz.
`data_vector`	Vector of numeric values. Time series vector of data.

Details

If time series is a vector of accelerometer data, then the outputted y-axis will have units of (acceleration^2)/Hz.

Explanations of some of the math: https://www.mathworks.com/help/signal/ug/power-spectral-density-estimates-using-fft.html

https://blog.endaq.com/why-the-power-spectral-density-psd-is-the-gold-standard-of-vibration-analysis

https://endaq.com/pages/power-spectral-density

https://medium.com/analytics-vidhya/breaking-down-confusions-over-fast-fourier-transform-fft-1561a029b1ab

Value

A List with two objects:

1. Vector of frequencies in Hz. The maximum frequency should be half the sampling frequency. Called Nyquist Frequency.
2. Vector of PSD values corresponding with each frequency. Units should be in the original units of the data vector squared and divided by frequency.

The vector of frequencies can be used as the x-axis values of a single sided spectrum amplitude plot. The vector of PSD values can be used as the y-axis values of the PSD plot.

Examples

```
#Create a vector of time that represent times where data are sampled.
Fs = 100; #sampling frequency in Hz
T = 1/Fs; #sampling period
L = 1000; #length of time vector
t = (0:(L-1))*T; #time vector

#Form a signal (time series) that contains two frequencies:
#1. 10 Hz with amplitude of 1
#2. 25 Hz with amplitude of 2
S <- 1*sin(2*pi*10*t) + 2*sin(2*pi*25*t);

results <- MakePowerSpectralDensity(Fs, S)

frequencies <- results[[1]]

PSD <- results[[2]]

#dev.new()
plot(frequencies, PSD, type = "l")
```

PSDDominantFrequencyForMultipleWindows

Calculate dominant frequency for multiple PSDs for a single frequency range

---

Description

Calculate dominant frequency for multiple PSDs for a single frequency range

Usage

```
PSDDominantFrequencyForMultipleWindows(
  list.of.windows,
  name.of.col.containing.time.series,
  sampling_frequency,
  x_start,
  x_end,
  x_increment
)
```

Arguments

list.of.windows	A list of windows (dataframes).
name.of.col.containing.time.series	A string that specifies the name of the column in the windows that correspond to the time series that should be used for making PSD.
sampling_frequency	Numeric value specifying sampling frequency in hertz. If data is sampled once every second, then sampling frequency is 1 Hz. If data is sampled once every 2 seconds, then sampling frequency is 0.5 Hz.
x_start	Numeric value specifying start of x value (frequency) to look at.
x_end	Numeric value specifying end of x value to look at.
x_increment	Numeric value specifying the increment of the x-values to use.

Value

A vector where each element is the dominant frequency of each window.

Examples

```
#Create a vector of time that represent times where data are sampled.
Fs = 100; #sampling frequency in Hz
T = 1/Fs; #sampling period
L = 1000; #length of time vector
t = (0:(L-1))*T; #time vector

#First signal
#1. 1 Hz with amplitude of 2
S1 <- 2*sin(2*pi*1*t)
level1.vals <- rep("a", length(S1))
level2.vals <- rep("1", length(S1))
S1.data.frame <- as.data.frame(cbind(t, S1, level1.vals, level2.vals))
colnames(S1.data.frame) <- c("Time", "Signal", "level1.ID", "level2.ID")
S1.data.frame[,"Signal"] <- as.numeric(S1.data.frame[,"Signal"])

#Second signal
```

```
#1. 1 Hz with amplitude of -4
#2. 2 Hz with amplitude of -2
S2 <- (-4)*sin(2*pi*1*t) - 2*sin(2*pi*2*t);
level1.vals <- rep("a", length(S2))
level2.vals <- rep("2", length(S2))
S2.data.frame <- as.data.frame(cbind(t, S2, level1.vals, level2.vals))
colnames(S2.data.frame) <- c("Time", "Signal", "level1.ID", "level2.ID")
S2.data.frame[,"Signal"] <- as.numeric(S2.data.frame[,"Signal"])

#Third signal
#1. 1 Hz with amplitude of 2
#2. 2 Hz with amplitude of 2
S3 <- 2*sin(2*pi*1*t) + 2*sin(2*pi*2*t);
level1.vals <- rep("a", length(S3))
level2.vals <- rep("3", length(S3))
S3.data.frame <- as.data.frame(cbind(t, S3, level1.vals, level2.vals))
colnames(S3.data.frame) <- c("Time", "Signal", "level1.ID", "level2.ID")
S3.data.frame[,"Signal"] <- as.numeric(S3.data.frame[,"Signal"])

#Fourth signal
#1. 1 Hz with amplitude of -2
S4 <- -2*sin(2*pi*1*t)
level1.vals <- rep("b", length(S4))
level2.vals <- rep("3", length(S4))
S4.data.frame <- as.data.frame(cbind(t, S4, level1.vals, level2.vals))
colnames(S4.data.frame) <- c("Time", "Signal", "level1.ID", "level2.ID")
S4.data.frame[,"Signal"] <- as.numeric(S4.data.frame[,"Signal"])

windows <- list(S1.data.frame, S2.data.frame, S3.data.frame, S4.data.frame)


results <- PSDDominantFrequencyForMultipleWindows(windows, "Signal", Fs, 0, 10, 0.01)
```

`PSDIdentifyDominantFrequency` *Given a time series vector, create a PSD and find the dominant frequency*

Description

The range to look for a dominant frequency (frequency corresponding to max PSD value) should be specified for this function. This function uses the MakePowerSpectralDensity() function and the IdentifyMaxOnXY() function together. If multiple equal maxima are found, then IdentifyMaxOnXY() will display a message.

Usage

```
PSDIdentifyDominantFrequency(
```

```
  sampling_frequency,
  data_vector,
  x_start = 0,
  x_end,
  x_increment
)
```

Arguments

sampling_frequency	Numeric value specifying sampling frequency in hertz. If data is sampled once every second, then sampling frequency is 1 Hz. If data is sampled once every 2 seconds, then sampling frequency is 0.5 Hz.
data_vector	Vector of numeric values. Timeseries vector of data.
x_start	Numeric value specifying start of x value to look at.
x_end	Numeric value specifying end of x value to look at.
x_increment	Numeric value specifying the increment of the x-values to use.

Value

A vector with two elements, The first element is the x value (frequency) where the max y value (PSD value) is found. The second element is the max y value.

Examples

```
#Create a vector of time that represent times where data are sampled.
Fs = 100; #sampling frequency in Hz
T = 1/Fs; #sampling period
L = 1000; #length of time vector
t = (0:(L-1))*T; #time vector

#First signal
#1. 1 Hz with amplitude of 2
S1 <- 2*sin(2*pi*1*t)
level1.vals <- rep("a", length(S1))
level2.vals <- rep("1", length(S1))
S1.data.frame <- as.data.frame(cbind(t, S1, level1.vals, level2.vals))
colnames(S1.data.frame) <- c("Time", "Signal", "level1.ID", "level2.ID")
S1.data.frame[,"Signal"] <- as.numeric(S1.data.frame[,"Signal"])


results <- PSDIdentifyDominantFrequency(Fs, S1.data.frame[,"Signal"], 0, 10, 0.01)
```

PSDIntegrationPerFreqBin *Given a time series vector, generate a PSD, then calculate integration for specified bins*

Description

Given a time series vector, generate a PSD, then calculate integration for specified bins

Usage

```
PSDIntegrationPerFreqBin(sampling_frequency, data_vector, frequency_bins)
```

Arguments

sampling_frequency	Numeric value specifying sampling frequency in hertz. If data is sampled once every second, then sampling frequency is 1 Hz. If data is sampled once every 2 seconds, then sampling frequency is 0.5 Hz.
data_vector	Vector of numeric values. Timeseries vector of data.
frequency_bins	A list of objects where each object is a vector with two elements. The first element is a numeric value specifying the start frequency of a bin. The second element is a numeric value specifying the end frequency of a bin. Each object corresponds to a new frequency bin for calculating integral. For integration, approxfun is used, so increment does not need to be specified.

Value

A list where each object is also a list. The nested list objects have the first element specifying the bin boundaries. The second element specifies the integral.

Examples

```
#Create a vector of time that represent times where data are sampled.
Fs = 100; #sampling frequency in Hz
T = 1/Fs; #sampling period
L = 1000; #length of time vector
t = (0:(L-1))*T; #time vector

#Form a signal (time series) that contains two frequencies:
#1. 10 Hz with amplitude of 1
#2. 25 Hz with amplitude of 2
S <- 1*sin(2*pi*10*t) + 2*sin(2*pi*25*t);

results <- MakePowerSpectralDensity(Fs, S)

frequencies <- results[[1]]
```

```
PSD <- results[[2]]

#dev.new()
plot(frequencies, PSD, type = "l")

bins <- list(
c(9, 11),
c(24,26),
c(9,26),
c(30,40)
)

integration.results <- PSDIntegrationPerFreqBin(Fs, S, bins)

message.captured <- list()

for(i in 1:length(integration.results)){

   message <- paste("Area in bin ", integration.results[[i]][[1]], " is ",
                    integration.results[[i]][[2]])

   message.captured[[i]] <- message

}
```

SingleBinPSDIntegrationForMultipleWindows
Calculate integral for multiple PSDs for a single frequency bin

---

Description

Calculate integral for multiple PSDs for a single frequency bin

Usage

```
SingleBinPSDIntegrationForMultipleWindows(
  list.of.windows,
  name.of.col.containing.time.series,
  sampling_frequency,
  single.bin.boundary
)
```

Arguments

`list.of.windows`
A list of windows (dataframes).

`name.of.col.containing.time.series`
: A string that specifies the name of the column in the windows that correspond to the time series that should be used for making PSD.

`sampling_frequency`
: Numeric value specifying sampling frequency in hertz. If data is sampled once every second, then sampling frequency is 1 Hz. If data is sampled once every 2 seconds, then sampling frequency is 0.5 Hz.

`single.bin.boundary`
: A numeric vector with two elements. First element is the start frequency for the bin. Second element is the end frequency of the bin. For integration, approxfun is used, so increment does not need to be specified.

Value

A vector where each element is the integration result of each window.

Examples

```
#Create a vector of time that represent times where data are sampled.
Fs = 100; #sampling frequency in Hz
T = 1/Fs; #sampling period
L = 1000; #length of time vector
t = (0:(L-1))*T; #time vector

#First signal
#1. 1 Hz with amplitude of 2
S1 <- 2*sin(2*pi*1*t)
level1.vals <- rep("a", length(S1))
level2.vals <- rep("1", length(S1))
S1.data.frame <- as.data.frame(cbind(t, S1, level1.vals, level2.vals))
colnames(S1.data.frame) <- c("Time", "Signal", "level1.ID", "level2.ID")
S1.data.frame[,"Signal"] <- as.numeric(S1.data.frame[,"Signal"])

#Second signal
#1. 1 Hz with amplitude of -4
#2. 2 Hz with amplitude of -2
S2 <- (-4)*sin(2*pi*1*t) - 2*sin(2*pi*2*t);
level1.vals <- rep("a", length(S2))
level2.vals <- rep("2", length(S2))
S2.data.frame <- as.data.frame(cbind(t, S2, level1.vals, level2.vals))
colnames(S2.data.frame) <- c("Time", "Signal", "level1.ID", "level2.ID")
S2.data.frame[,"Signal"] <- as.numeric(S2.data.frame[,"Signal"])

#Third signal
#1. 1 Hz with amplitude of 2
#2. 2 Hz with amplitude of 2
S3 <- 2*sin(2*pi*1*t) + 2*sin(2*pi*2*t);
level1.vals <- rep("a", length(S3))
level2.vals <- rep("3", length(S3))
S3.data.frame <- as.data.frame(cbind(t, S3, level1.vals, level2.vals))
```

```
colnames(S3.data.frame) <- c("Time", "Signal", "level1.ID", "level2.ID")
S3.data.frame[,"Signal"] <- as.numeric(S3.data.frame[,"Signal"])

#Fourth signal
#1. 1 Hz with amplitude of -2
S4 <- -2*sin(2*pi*1*t)
level1.vals <- rep("b", length(S4))
level2.vals <- rep("3", length(S4))
S4.data.frame <- as.data.frame(cbind(t, S4, level1.vals, level2.vals))
colnames(S4.data.frame) <- c("Time", "Signal", "level1.ID", "level2.ID")
S4.data.frame[,"Signal"] <- as.numeric(S4.data.frame[,"Signal"])

windows <- list(S1.data.frame, S2.data.frame, S3.data.frame, S4.data.frame)


results <- SingleBinPSDIntegrationForMultipleWindows(windows, "Signal", Fs, c(0,2))
```

`SingleBinPSDIntegrationOrDominantFreqComparison`

Given sets of windows (dataframes) corresponding to different combos, see if the integration or dominant frequency of a specific frequency range is significantly different between the combos

---

Description

Just for a single frequency bin: For Each combination in level.combinations, generate the integral or dominant frequency for each window of each combination. At this point, we should have vectors of integrals or dominant frequency with each vector corresponding to a different combo. Now we want to see if the integrals or dominant frequencies in each combo significantly differ from the other combos. Kruskal-Wallis test is used as a non-parametric ANOVA test to see if the combos have integrals or dominant frequencies that are significantly different.

Usage

```
SingleBinPSDIntegrationOrDominantFreqComparison(
  list.of.windows,
  name.of.col.containing.time.series,
  level1.column.name,
  level2.column.name,
  level.combinations,
  level.combinations.labels,
  sampling_frequency,
  single.bin.boundary = NULL,
  x_start = NULL,
  x_end = NULL,
  x_increment = NULL,
  integration.or.dominant.freq
)
```

Arguments

`list.of.windows`
: A list of windows (dataframes).

`name.of.col.containing.time.series`
: A string that specifies the name of the column in the windows that correspond to the time series that should be used for making PSD.

`level1.column.name`
: A String that specifies the column name to use for the first level. This column should only contain one unique value within each window.

`level2.column.name`
: A String that specifies the column name to use for the second level. This column should only contain one unique value within each window.

`level.combinations`
: A List containing Lists. Each list that it contains has two vectors. The first vector specifying the values for level1 and the second vector specifying the values for level2. Each list element will correspond to a new line on the plot.

`level.combinations.labels`
: A vector of strings that labels each combination. This is used for naming the groups in integrals.with.combo.labels

`sampling_frequency`
: Numeric value specifying sampling frequency in hertz. If data is sampled once every second, then sampling frequency is 1 Hz. If data is sampled once every 2 seconds, then sampling frequency is 0.5 Hz.

`single.bin.boundary`
: A numeric vector with two elements. First element is the start frequency for the bin. Second element is the end frequency of the bin.

`x_start`
: Numeric value specifying start of the new x-axis for the averaged PSD. Default is 0 Hz.

`x_end`
: Numeric value specifying end of the new x-axis for the averaged PSD. Maximum value is the sampling_frequency divided by 2.

`x_increment`
: Numeric value specifying increment of the new x-axis for the averaged PSD.

`integration.or.dominant.freq`
: A string with two possible values for choosing whether integral or dominant frequency should be calculated and compared: "integration" or "dominant_freq".

Details

Need to specify whether to compare integral or dominant frequency: If integral (total power in frequency bin) is the value to compare, then SingleBinPSDIntegrationForMultipleWindows() is used. If dominant frequency (frequency corresponding to max PSD value in frequency bin) is the value to compare, then PSDDominantFrequencyForMultipleWindows() is used.

Value

A list with 3 objects:

1. integrals.with.combo.labels: Dataframe used for statistical testing.

2. kruskal.test.res: Results from Kruskal-Willis testing.
3. pairwise.wilcox.rest.res: Results from pairwise Wilcoxo testing

Examples

```
#Create a vector of time that represent times where data are sampled.
Fs = 100; #sampling frequency in Hz
T = 1/Fs; #sampling period
L = 1000; #length of time vector
t = (0:(L-1))*T; #time vector

#First signal
#1. 1 Hz with amplitude of 2
S1 <- 2*sin(2*pi*1*t)
level1.vals <- rep("a", length(S1))
level2.vals <- rep("1", length(S1))
S1.data.frame <- as.data.frame(cbind(t, S1, level1.vals, level2.vals))
colnames(S1.data.frame) <- c("Time", "Signal", "level1.ID", "level2.ID")
S1.data.frame[,"Signal"] <- as.numeric(S1.data.frame[,"Signal"])

#Second signal
#1. 1 Hz with amplitude of -4
#2. 2 Hz with amplitude of -2
S2 <- (-4)*sin(2*pi*1*t) - 2*sin(2*pi*2*t);
level1.vals <- rep("a", length(S2))
level2.vals <- rep("2", length(S2))
S2.data.frame <- as.data.frame(cbind(t, S2, level1.vals, level2.vals))
colnames(S2.data.frame) <- c("Time", "Signal", "level1.ID", "level2.ID")
S2.data.frame[,"Signal"] <- as.numeric(S2.data.frame[,"Signal"])

#Third signal
#1. 1 Hz with amplitude of 2
#2. 2 Hz with amplitude of 2
S3 <- 2*sin(2*pi*1*t) + 2*sin(2*pi*2*t);
level1.vals <- rep("a", length(S3))
level2.vals <- rep("3", length(S3))
S3.data.frame <- as.data.frame(cbind(t, S3, level1.vals, level2.vals))
colnames(S3.data.frame) <- c("Time", "Signal", "level1.ID", "level2.ID")
S3.data.frame[,"Signal"] <- as.numeric(S3.data.frame[,"Signal"])

#Fourth signal
#1. 1 Hz with amplitude of -2
S4 <- -2*sin(2*pi*1*t)
level1.vals <- rep("b", length(S4))
level2.vals <- rep("3", length(S4))
S4.data.frame <- as.data.frame(cbind(t, S4, level1.vals, level2.vals))
colnames(S4.data.frame) <- c("Time", "Signal", "level1.ID", "level2.ID")
S4.data.frame[,"Signal"] <- as.numeric(S4.data.frame[,"Signal"])

#Fifth signal
```

```
#1. 5 Hz with amplitude of -2
S5 <- -2*sin(2*pi*5*t)
level1.vals <- rep("c", length(S5))
level2.vals <- rep("1", length(S5))
S5.data.frame <- as.data.frame(cbind(t, S5, level1.vals, level2.vals))
colnames(S5.data.frame) <- c("Time", "Signal", "level1.ID", "level2.ID")
S5.data.frame[,"Signal"] <- as.numeric(S5.data.frame[,"Signal"])

#Extra representation of S2 dataframe to show an example that has enough samples
#to have significance for Kruskal-Wallis test
windows <- list(S1.data.frame, S2.data.frame, S2.data.frame, S2.data.frame, S2.data.frame,
S2.data.frame, S2.data.frame, S2.data.frame, S2.data.frame, S2.data.frame, S3.data.frame,
S4.data.frame,
S5.data.frame, S5.data.frame, S5.data.frame, S5.data.frame, S5.data.frame)

#Gets the composite of the first, second, and third signal. Should result in a flat signal.
FirstComboToUse <- list( c("a"), c(1, 2, 3) )

#Gets the composite of the third and fourth signal
SecondComboToUse <- list( c("a", "b"), c(3) )

#Gets the composite of fifth signal
ThirdComboToUse <- list( c("c"), c(1) )


#PSD--------------------------------------------------------------------------

PSD.results <- AutomatedCompositePlotting(list.of.windows = windows,
                          name.of.col.containing.time.series = "Signal",
                          x_start = 0,
                          x_end = 10,
                          x_increment = 0.01,
                          level1.column.name = "level1.ID",
                          level2.column.name = "level2.ID",
                          level.combinations = list(FirstComboToUse,
                                                    SecondComboToUse,
                                                    ThirdComboToUse),
                          level.combinations.labels = c("Signal 1 + 2 + 3",
                                                        "Signal 3 + 4",
                                                        "Signal 5"),
                          plot.title = "Example",
                          plot.xlab = "Hz",
                          plot.ylab = "(Original units)^2/Hz",
                          combination.index.for.envelope = 2,
                          TimeSeries.PSD.LogPSD = "PSD",
                          sampling_frequency = 100)

ggplot.obj.PSD <- PSD.results[[2]]

#Integration------------------------------------------------------------------------

#Compare integration for the 1.5-2.5 Hz bin. P-value should not indicate
#significant difference
```

```
integration.compare.res <- SingleBinPSDIntegrationOrDominantFreqComparison(
list.of.windows = windows,
name.of.col.containing.time.series = "Signal",
level1.column.name = "level1.ID",
level2.column.name = "level2.ID",
level.combinations = list(FirstComboToUse, SecondComboToUse),
level.combinations.labels = c("Signal 1 + 2 + 3", "Signal 3 + 4"),
sampling_frequency = 100,
single.bin.boundary = c(1.5, 2.5),
integration.or.dominant.freq = "integration")

#Kruskal-Wallis test results
integration.compare.res[[2]]

#Compare integration for the 0.5-1.5 Hz bin. P-value should indicate
#significant difference
integration.compare.res2 <- SingleBinPSDIntegrationOrDominantFreqComparison(
list.of.windows = windows,
name.of.col.containing.time.series = "Signal",
level1.column.name = "level1.ID",
level2.column.name = "level2.ID",
level.combinations = list(FirstComboToUse, SecondComboToUse),
level.combinations.labels = c("Signal 1 + 2 + 3", "Signal 3 + 4"),
sampling_frequency = 100,
single.bin.boundary = c(0.5,1.5),
integration.or.dominant.freq = "integration")

#Kruskal-Wallis test results
integration.compare.res2[[2]]



#Dominant Frequency-----------------------------------------------------------------------

#Compare dominant frequency P-value should not indicate
#significant difference
integration.compare.res3 <- SingleBinPSDIntegrationOrDominantFreqComparison(
list.of.windows = windows,
name.of.col.containing.time.series = "Signal",
level1.column.name = "level1.ID",
level2.column.name = "level2.ID",
level.combinations = list(FirstComboToUse, SecondComboToUse),
level.combinations.labels = c("Signal 1 + 2 + 3", "Signal 3 + 4"),
sampling_frequency = 100,
x_start = 0,
x_end = 10,
x_increment = 0.01,
integration.or.dominant.freq = "dominant_freq")

#Kruskal-Wallis test results
integration.compare.res3[[2]]
```

```
#Compare dominant frequency P-value should indicate
#significant difference
integration.compare.res4 <- SingleBinPSDIntegrationOrDominantFreqComparison(
list.of.windows = windows,
name.of.col.containing.time.series = "Signal",
level1.column.name = "level1.ID",
level2.column.name = "level2.ID",
level.combinations = list(SecondComboToUse, ThirdComboToUse),
level.combinations.labels = c("Signal 3 + 4", "Signal 5"),
sampling_frequency = 100,
x_start = 0,
x_end = 10,
x_increment = 0.01,
integration.or.dominant.freq = "dominant_freq")

#Kruskal-Wallis test results
integration.compare.res4[[2]]
#Values used in comparison of the two groups
integration.compare.res4[[1]]
```

Index