Package ‘sonicscrewdriver’

November 19, 2019

Title Bioacoustic Analysis and Publication Tools
Version 0.0.1
Description Provides basic tools for manipulating sound files for bioacoustic analysis, and preparing analyses these for publication. The package validates that values are physically possible wherever feasible.
Depends R (>= 3.4.0)
Imports tuneR, seewave, methods, ggplot2
License GPL-3
Encoding UTF-8
LazyData true
RoxygenNote 6.1.1
Language en-GB
Suggests testthat, covr, WaveletComp, devtools, googleCloudStorageR, googleLanguageR
NeedsCompilation no
Author Baker Ed [aut, cre],
Geissman Quentin [ctb]
Maintainer Baker Ed <ed@ebaker.me.uk>
Repository CRAN
Date/Publication 2019-11-19 13:20:02 UTC

R topics documented:

addSpectra ................................................................. 2
autoBandPass .................................................................. 3
beatComplexity .............................................................. 4
beatSpectrum ................................................................. 5
convert2Celsius ............................................................. 6
convert2Fahrenheit ....................................................... 6
convert2Kelvin .............................................................. 7
convert2Pascals ............................................................ 7
addSpectra

Add two spectra from seewave

Description

This function takes two spectra from seewave (or equivalent) and adds their values. The spectra must have the same bins.

Usage

addSpectra(s1, s2, coerceNegative = "no")
autoBandPass

**Arguments**

- s1: First spectrum
- s2: Second spectrum
- coerceNegative: Sets any values below zero to zero, accepted values "input", "output" or "both".

**Value**

A spectrum of s1+s2

**Examples**

```r
## Not run:
addSpectra(spec1, spec2)
addSpectra(spec1, spec2, coerceNegative="input")
## End(Not run)
```

---

**autoBandPass**  
**Automatic Band Pass Filter**

**Description**

Creates an automatic bandpass filter based on the strongest frequency. The allowed bandwidth can be an integer multiple of the bandwidth at either -3dB or -10dB.

**Usage**

```r
autoBandPass(wave, bw = "-3dB", n.bw = 1, lowcut = 1000)
```

**Arguments**

- wave: A Wave object
- bw: Either -3dB or -10dB. This is calculated by frequencyStats
- n.bw: The number of bandwidths either side of the centre of the centre to keep
- lowcut: High-pass filtering is applied at this frequency before calculating the centre frequency and bandwidth

**Value**

A band-pass filtered Wave object
Examples

```r
## Not run:
autoBandPass(sheep)
autoBandPass(sheep, bw="-3dB", n.bw=1, lowcut=1000)
autoBandPass(sheep, bw="-10dB", n.bw=2, lowcut=0)

## End(Not run)
```

---

**beatComplexity**

*Beat spectrum complexity*

Description

This function computes a `beatSpectrum` and calculates some basic measurements of its complexity. The complexity value is calculated as the maximum identified repeating period (in seconds) divided by the number of peaks.

Usage

```r
beatComplexity(wave, plot = FALSE)
```

Arguments

- `wave` A Wave object
- `plot` If TRUE a spectrogram overlaid with the peaks is plotted.

Value

A list of the complexity, a vector of the peak periods, and the number of peaks.

Examples

```r
## Not run:
beatComplexity(sheep)
beatComplexity(sheep, plot=TRUE)

## End(Not run)
```
beatSpectrum

Computes a beat spectrum

Description

Beat spectra represent the periodicity in signal amplitude. It is computed by performing a continuous wavelet transform on the envelope of a preprocessed signal, and processing the average power per frequency band.

Usage

beatSpectrum(wave, min_period = 0.005, max_period = 30, dj = 1/32, ...)

Arguments

wave an R object or path to a wave file
min_period the minimal rhythmicity period expected, in seconds
max_period the maximal rhythmicity period expected, in seconds
dj the frequency resolution of the cwt (in voices per octave)
... extra arguments passed to analyze.wavelet()

Value

a spectrum as a data frame. It contains two columns: power and period. The number of rows depend on the resolution and frequency range.

Author(s)

Quentin Geissmann

Examples

## Not run:
beatSpectrum(sheep)
beatSpectrum(sheep, min_period=0.005, max_period=30, dj=1/32)

## End(Not run)
**convert2Celsius**  
*Convert temperature to Celsius*

**Description**
Converts temperature measurements into Celsius

**Usage**
```
convert2Celsius(temp, input = "K")
```

**Arguments**
- `temp`: The value of the temperature to convert
- `input`: The unit of the temperature to convert, allowed values are "K", "F".

**Value**
Numeric value in degrees Celsius

**Examples**
```
convert2Celsius(15, input="K")
convert2Celsius(15, input="F")
```

---

**convert2Fahrenheit**  
*Convert temperature to Fahrenheit*

**Description**
Converts temperature measurements into Fahrenheit

**Usage**
```
convert2Fahrenheit(temp, input)
```

**Arguments**
- `temp`: The value of the temperature to convert
- `input`: The unit of the temperature to convert, allowed values are "K", "C".
### convert2Kelvin

Convert temperature to Kelvin

#### Description
Converting temperature measurements into Kelvin

#### Usage

```r
classify2Kelvin(temp, input = "C")
```

#### Arguments
- **temp**
  - The value of the temperature to convert

- **input**
  - The unit of the temperature to convert, allowed values are "C", "F".

#### Examples

```r
classify2Kelvin(15, input="C")
classify2Kelvin(15, input="F")
```

---

### convert2Pascals

Convert pressure to Pascals

#### Description
Converting pressure measurements into Pascals

#### Usage

```r
classify2Pascals(P, input = "kPa")
```
Arguments

P  The value of the pressure to convert
input  The unit of the pressure to convert, allowed values are "kPa".

Value

The numeric value in Pascals

Examples

convert2Pascals(1, input="kPa")

____
cutws  Cut wave by samples

Description

Extract a section of a Wave object based on sample positions

Usage

cutws(wave, from, to, plot = FALSE)

Arguments

wave  A Wave object
from  First sample to return
to  Last sample to return
plot  If TRUE shows the cut region within the original waveform

Value

A Wave object

Examples

## Not run:
cutws(sheep, 1, 20)
cutws(sheep, 1, 20, plot=TRUE)

## End(Not run)
**data2Wave**

*Convert data into a Wave object*

**Description**

Make a sequence of data into a normalised Wave object.

**Usage**

```r
data2Wave(left, samp.rate = 44100, bit = 16)
```

**Arguments**

- `left`: Data for audio channel
- `samp.rate`: Sampling rate for Wave object
- `bit`: Bit depth of Wave object

**Value**

A mono Wave object.

**Examples**

```r
pattern <- seq(from=-1, to=1, length.out=100)
data <- rep.int(pattern, 100)
w <- data2Wave(data)
```

---

**defaultCluster**

*Create Default Cluster for Windowing*

**Description**

Creates a default cluster using one less than the total cores available on the system. By default this uses forking, which may not be available on `Windows`.

**Usage**

```r
defaultCluster(fork = TRUE)
```

**Arguments**

- `fork`: If TRUE uses forking to create the cluster

**Value**

A cluster object for parallel processing
Examples

## Not run:
cl <- defaultCluster()
stopCluster(cl)
cl <- defaultCluster(FALSE)
stopCluster(cl)

## End(Not run)

<table>
<thead>
<tr>
<th>dutyCycle</th>
<th>Calculate the duty cycle of a wave</th>
</tr>
</thead>
</table>

Description

Proportion of a wave with signal above the limit

Usage

dutyCycle(wave, limit = 0.1, output = "unit")

Arguments

- wave: A Wave object
- limit: Threshold above which to consider the signal
- output: If "unit" the duty cycle will be in the range 0-1. For a percentage use "percent".

Value

A numerical value for the duty cycle between 0 and 1 (or 0 and 100)

Examples

wave <- tuneR::sine(2000)
dc <- dutyCycle(wave)
pc <- dutyCycle(wave, output="percent")
**entropyStats**

Various measurements of frequency values for a Wave object

**Description**
Calculates the peak, centre, bandwidth and quality factor. The quality factor (Q) is calculated at both -3dB and -10dB as discussed by Bennett-Clark (1999) <doi:10.1080/09524622.1999.9753408>.

**Usage**

```r
entropyStats(wave)
```

**Arguments**

- `wave`  
  A Wave object

**Value**

A list of spectral entropy types.

**Examples**

```r
## Not run:  
entropyStats(sheep)
## End(Not run)
```

**frequencySound**

Get the frequency from wavelength and speed of sound

**Description**
Calculates the frequency of a sound wave given the wavelength and speed of sound in that medium.

**Usage**

```r
frequencySound(wl, s)
```

**Arguments**

- `wl`  
  Wavelength
- `s`  
  Speed of sound

**Value**

Frequency of the sound in Hertz
Examples

```r
f <- frequencySound(wl=100, s=343)
```

---

**frequencyStats**  
**Various measurements of frequency values for a Wave object**

**Description**

Calculates the peak, centre, bandwidth and quality factor. The quality factor (Q) is calculated at both -3dB and -10dB as discussed by Bennett-Clark (1999) [doi: 10.1080/09524622.1999.9753408].

**Usage**

```r
frequencyStats(wave, wave_spec = NULL, warn = TRUE, lowcut = 1, plot = FALSE)
```

**Arguments**

- `wave`: A Wave object
- `wave_spec`: A precomputed spectrum (optional, if not present will be generated)
- `warn`: If TRUE provides warnings when values are not consistent
- `lowcut`: Frequency (in kHz) values below which are ignored.
- `plot`: IF TRUE displays values

---

**generateNoise**  
**Add noise to a soundwave**

**Description**

Adding noise to a soundwave allows for testing of the robustness of automated identification algorithms to noise.

**Usage**

```r
generateNoise(wave, noise = c("white"), noiseAdd = FALSE, noiseRatio = 0.5, output = "file", plot = FALSE)
```

**Arguments**

- `wave`: Wave file to add noise to
- `noise`: Vector of noise to add (unif, gaussian, white, pink, power, red, frequency of a sine wave in Hz, or filename)
- `noiseAdd`: If TRUE all noise sources are added to wave. If FALSE separate outputs are created for each noise source.
- `noiseRatio`: Ratio of maximum noise amplitude to the maximum amplitude in wave
- `output`: TODO: Is this implemented?
- `plot`: If TRUE various plots are made to show how noise is added.
**gs_transcribe**

**Value**

A list of Wave objects with the required noise added.

---

**gs_transcribe**  
*Google Speech API Transcribe*

**Description**

Wrapper around various Google packages to simplify speech transcription.

**Usage**

```r
gs_transcribe(filename, bucket = NULL, ...)
```

**Arguments**

- `filename`: Path to file for analysis
- `bucket`: Storage bucket on Google Cloud for larger files
- `...`: Additional arguments to pass to `gl_speech()`

**Value**

A `gs_transcribe` object containing details of the transcription

**Examples**

```r
## Not run:
gs_transcribe("demo.wav")
## End(Not run)
```

---

**labelPadding**  
*Pad labels with interval*

**Description**

Takes labels from Google Speech API transcript and pads the time by a specified number of seconds.

**Usage**

```r
labelPadding(t, pad = 0.5, max_t = NULL)
```
labelReduction

Arguments

- **t**: Transcript from Google Speech API
- **pad**: Amount of time (in seconds) to add to start and end
- **max_t**: Optional. The duration of the file, so padding does not exceed length of file.

Value

A modified Google Speech API transcript object

Examples

```r
## Not run:
labelPadding(t, pad=2, max_t=duration(wave))

## End(Not run)
```

labelReduction  

Combines labels which overlap into single continuous regions

Description

Takes labels from Google Speech API transcript and combines overlapping labels.

Usage

```r
labelReduction(t)
```

Arguments

- **t**: Transcript from Google Speech API

Value

A list containing start and end times of speech containing regions

Examples

```r
## Not run:
labelReduction(t)

## End(Not run)
```
ntd

Natural Time Domain

Description

Runs a function on the wave and outputs values in the Natural Time Domain (see Varotsos, Sarlis & Skordas(2011) <doi:10.1007/978-3-642-16449-1>).

Usage

ntd(wave, events, FUN, normalise = FALSE, argument = "wave", ...)

Arguments

- **wave**: A Wave object containing pulses
- **events**: Onset of detected events, e.g. from pulseDetection()
- **FUN**: The function to run
- **normalise**: If TRUE the output is a probability density
- **argument**: If "wave" supplies a weave object to the function, if "vector" supplies the left channel as a numeric vector.
- **...**: Additional arguments to FUN

Value

A list of outputs from the applied function

---

parseFilename

Parse a filename

Description

Attempts to extract meaningful information from a filename.

Usage

parseFilename(string)

Arguments

- **string**: A filename

Value

A list of raw results, plus calculated values for date, time and device.
Description


Usage

\[
\text{pd\_dietrich2004(} \text{wave, } U = 120, \text{ gamma = 0.05, alpha = 1.4, scaling = 32, } V = 480, \text{ psi = 1) }
\]

Arguments

\[
\begin{array}{ll}
\text{wave} & \text{A Wave object} \\
U & \text{Window length} \\
\text{gamma} & \text{Gamma} \\
\text{alpha} & \text{Alpha} \\
\text{scaling} & \text{Scaling} \\
V & \text{V Window length} \\
\text{psi} & \text{Psi}
\end{array}
\]

Value

A list of input values plus the onset and offset times of pulses

Description

Detects pulses in a Wave.

Usage

\[
\text{pd\_simple(} \text{wave, } U = 120, \text{ gamma = 0.05, alpha = 1.4, scaling = 32, } V = 480, \text{ psi = 1) }
\]
**pulseDetection**

**Arguments**

- **wave**: A Wave object
- **U**: Window length
- **gamma**: Gamma
- **alpha**: Alpha
- **scaling**: Scaling
- **V**: V Window length
- **psi**: Psi

**Description**

Detects pulses in a Wave, defaults to using Dietrich (2004).

**Usage**

```r
pulseDetection(wave, method = "simple", ...)
```

**Arguments**

- **wave**: A Wave object containing pulses
- **method**: Which method to use for pulse detection
- **...**: Other arguments to pass to pulse detection function

---

**pulseIntervals**

**Pulse intervals**

**Description**

Used to locate area of no pulses from the results of pulseDetection().

**Usage**

```r
pulseIntervals(pulses, nsd = 2)
```

**Arguments**

- **pulses**: The result of a pulseDetection.
- **nsd**: The number of standard deviations each sid of the mean pulse interval to discard

**Value**

A list of onset and offset times for pulses
**rainfallDetection**  
**Rainfall detection**

**Description**

**Usage**

```
rainfallDetection(wave, method = "bedoya2017", ...)```

**Arguments**

- `wave`: A Wave object to detect rainfall in
- `method`: Which rainfall detection method to use ("bedoya2017", "hardRain")
- `...`: Other arguments to pass to rain detection function

**Value**
Numeric value from the rainfall detection algorithm chosen.

**Examples**

```
## Not run:
rainfallDetection(sheep, method="bedoya2017")
rainfallDetection(sheep, method="hardRain")

## End(Not run)
```

---

**sDuration**  
**Sample duration**

**Description**
Calculates the time represented by n samples in a Wave.

**Usage**

```
sDuration(n = 1, wave = NULL, samp.rate = NULL)```

**Arguments**

- `n`: The number of the samples
- `wave`: A Wave object containing pulses
- `samp.rate`: Integer sampling rate
sheepFrequencyStats

Value
A numeric value in seconds

Examples

sDuration(n=20, samp.rate=44100)
## Not run:
sDuration(n=20, wave=sheep)#'
## End(Not run)

sheepFrequencyStats  Sheep frequencyStats

Description
The frequencyStats of the sheep data file from the seewave package.

Usage
sheepFrequencyStats

Format
An object of class list of length 3.

soundSpeed  Calculate the speed of sound in a medium

Description
Given sufficient parameters (i.e. wavelength and frequency, bulk modulus and density) this function calculates the speed of sound.

Usage

soundSpeed(wl = NULL, f = NULL, bulkModulus = NULL, density = NULL)

Arguments

wl  Wavelength
f  Frequency
bulkModulus  Bulk modulus
density  Density
**soundSpeedMedium**  
*Get the speed of sound in a medium*

**Description**
Provides typical values of the speed of sound in a given medium (air, sea water, freshwater).

**Usage**
soundSpeedMedium(medium = "air")

**Arguments**
- medium: Propagation medium (default is "air")

**Value**
Typical value of the speed of sound in m/s for the medium

**Examples**
soundSpeedMedium("air")
soundSpeedMedium("sea water")

---

**soundSpeed_cramer1993**  
*Speed of sound in air using Cramer (1993)*

**Description**
Calculate the speed of sound in air using the method described in Cramer (1993) <doi:10.1121/1.405827>

**Usage**
soundSpeed_cramer1993(temp, temp.unit = "C", pressure, pressure.unit = "kPa", RH, MoleFracCO2 = 400^-6)

**Arguments**
- temp: Temperature
- temp.unit: Temperature unit
- pressure: Pressure
- pressure.unit: Pressure unit
- RH: Relative humidity
- MoleFracCO2: Mole fraction of CO2
**specStats**

**Value**

Numeric value of the speed of sound in m/s

**Examples**

```r
soundSpeed_cramer1993(14, pressure=3, RH=10)
soundSpeed_cramer1993(14, temp.unit="C", pressure=3, pressure.unit="kPa", RH=10)
```

---

**specStats**

*Calculate and plot statistics on a frequency spectrum*

**Description**

Given a list of outputs from meanspec generates a plot with the mean shown by a line, and either the minimum/maximum values or one standard deviation shown by a ribbon.

**Usage**

```r
specStats(spectra, stats = "minMax", line.col = "black", ribbon.col = "grey70")
```

**Arguments**

- `spectra`: A list of spectra
- `stats`: Either minMax or sd
- `line.col`: Colour for the line
- `ribbon.col`: Colour for the ribbon

**Value**

A ggplot2 object

---

**ste**

*Short term energy*

**Description**

Computes the short term energy of a Wave.

**Usage**

```r
ste(wave, method = "dietrich2004", ...)
```
subtractSpectra

Arguments

- `wave` A Wave object
- `...` Other arguments to pass to STE function

Value

A vector of short term energy values

Examples

```r
## Not run:
ste(sheep, method="dietrich2004")
## End(Not run)
```

STP

`STP`: Standard Temperature and Pressure

Description

Dataset compiled from various sources for differing values of STP.

Usage

`STP`

Format

An object of class `list` of length 2.

subtractSpectra

`Subtract two spectra from seewave`

Description

This function takes two spectra from seewave (or equivalent) and subtracts their values. The spectra must have the same bins.

Usage

```r
subtractSpectra(s1, s2, coerceNegative = "no")
```
**tSamples**

Arguments

- s1: First spectrum
- s2: Second spectrum
- coerceNegative: Sets any values below zero to zero, accepted values "input", "output" or "both".

Value

A spectrum of s1 - s2

Examples

```r
## Not run:
subtractSpectra(spec1, spec2)
subtractSpectra(spec1, spec2, coerceNegative="both")

## End(Not run)
```

---

**tSamples**  
*Samples per time period*

Description

Calculates the number of samples for a given duration of a wave

Usage

tSamples(time = 1, wave = NULL, samp.rate = NULL)

Arguments

- time: The duration in seconds
- wave: A Wave object containing pulses
- samp.rate: Integer sampling rate

Value

Number of samples

Examples

tSamples(10, samp.rate=44100)

## Not run:
tSamples(10, wave=sheep)

## End(Not run)
validateIsWave  
*Check an object is a Wave object*

**Description**

Helper function to test that the input is a Wave object. Will create an error if not.

**Usage**

```r
validateIsWave(wave)
```

**Arguments**

- **wave**: Object to test

---

windowing  
*Windowing Function for Wave Objects*

**Description**

Separates a Wave object into windows of a defined length and runs a function on the window section. Windows may overlap, and the function can make use of ‘parallel’ package for multicore processing.

**Usage**

```r
windowing(wave, window.length, window.overlap = 0, bind.wave = TRUE, 
FUN, ..., cluster = NULL)
```

**Arguments**

- **wave**: A Wave object
- **window.length**: The lag used to create the A-matrix
- **window.overlap**: A matrix used to code the Duration-Shape pairs
- **bind.wave**: If TRUE and FUN returns wave objects these are combined into a single object
- **FUN**: If TRUE plots the workings of the coding algorithm
- **...**: Additional parameters to FUN
- **cluster**: A cluster form the ‘parallel’ package for multicore computation

**Examples**

```r
## Not run:
windowing(wave, window.length=1000, window.overlap=0, bind.wave=TRUE, FUN=noChange)
```

```r
## End(Not run)
```
zeroSpectrum

### Description
This function takes a spectrum from seewave and creates a new zero-valued spectrum with the same structure.

### Usage
zeroSpectrum(s1)

### Arguments
- **s1**
  Spectrum to emulate the structure of.

### Value
A zero-valued spectrum.

### Examples
```r
# Not run:
zeeroSpectrum(spec)
# End(Not run)
```
Index

*Topic datasets
  sheepFrequencyStats, 19
  STP, 22
*Topic wave
  defaultCluster, 9
  windowing, 24
addSpectra, 2
autoBandPass, 3
beatComplexity, 4
beatSpectrum, 5
convert2Celsius, 6
convert2Fahrenheit, 6
convert2Kelvin, 7
convert2Pascals, 7
cutws, 8
data2Wave, 9
defaultCluster, 9
dutyCycle, 10
entropyStats, 11
frequencySound, 11
frequencyStats, 12
generateNoise, 12
gs_transcribe, 13
labelPadding, 13
labelReduction, 14
ntd, 15
parseFilename, 15
pd_dietrich2004, 16
pd_simple, 16
pulseDetection, 17
pulseIntervals, 17
rainfallDetection, 18
sDuration, 18
sheepFrequencyStats, 19
soundSpeed, 19
soundSpeed_cramer1993, 20
soundSpeedMedium, 20
specStats, 21
ste, 21
STP, 22
subtractSpectra, 22
tSamples, 23
validateIsWave, 24
windowing, 24
zeroSpectrum, 25