Package ‘sonicscrewdriver’

October 14, 2022

Title Bioacoustic Analysis and Publication Tools
Version 0.0.4
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, jsonlite, mime, suncalc, hms
License GPL-3
Encoding UTF-8
LazyData true
RoxygenNote 7.1.1
Language en-GB
Suggests av, testthat, covr, WaveletComp, devtools,
googlcCloudStorageR, googleLanguageR, knitr, rmarkdown,
spelling
VignetteBuilder knitr
NeedsCompilation no
Author Baker Ed [aut, cre],
Geissman Quentin [ctb]
Maintainer Baker Ed <ed@ebaker.me.uk>
Repository CRAN
Date/Publication 2021-05-02 15:40:02 UTC

R topics documented:

ab_annotations ......................................................... 3
ab_seqss_nearestStart ............................................. 4
addSpectra ............................................................. 4
audiomoth_config .................................................... 5
audiomoth_wave ...................................................... 6
<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>audio_filesize</td>
<td>6</td>
</tr>
<tr>
<td>autoBandPass</td>
<td>7</td>
</tr>
<tr>
<td>beatComplexity</td>
<td>8</td>
</tr>
<tr>
<td>beatSpectrum</td>
<td>8</td>
</tr>
<tr>
<td>convert2bytes</td>
<td>9</td>
</tr>
<tr>
<td>convert2Celsius</td>
<td>10</td>
</tr>
<tr>
<td>convert2dyne_cm2</td>
<td>10</td>
</tr>
<tr>
<td>convert2Fahrenheit</td>
<td>11</td>
</tr>
<tr>
<td>convert2Kelvin</td>
<td>11</td>
</tr>
<tr>
<td>convert2Pascals</td>
<td>12</td>
</tr>
<tr>
<td>convert2seconds</td>
<td>12</td>
</tr>
<tr>
<td>cutws</td>
<td>13</td>
</tr>
<tr>
<td>data2Wave</td>
<td>14</td>
</tr>
<tr>
<td>dayPhase</td>
<td>14</td>
</tr>
<tr>
<td>dayPhases</td>
<td>15</td>
</tr>
<tr>
<td>daysPhases</td>
<td>16</td>
</tr>
<tr>
<td>defaultCluster</td>
<td>16</td>
</tr>
<tr>
<td>dutyCycle</td>
<td>17</td>
</tr>
<tr>
<td>entropyStats</td>
<td>18</td>
</tr>
<tr>
<td>frequencySound</td>
<td>18</td>
</tr>
<tr>
<td>frequencyStats</td>
<td>19</td>
</tr>
<tr>
<td>generateNoise</td>
<td>19</td>
</tr>
<tr>
<td>gs_transcribe</td>
<td>20</td>
</tr>
<tr>
<td>jitter</td>
<td>21</td>
</tr>
<tr>
<td>labelPadding</td>
<td>21</td>
</tr>
<tr>
<td>labelReduction</td>
<td>22</td>
</tr>
<tr>
<td>naturalFrequency</td>
<td>23</td>
</tr>
<tr>
<td>ntd</td>
<td>23</td>
</tr>
<tr>
<td>parseFilename</td>
<td>24</td>
</tr>
<tr>
<td>pd_dietrich2004</td>
<td>25</td>
</tr>
<tr>
<td>pd_simple</td>
<td>25</td>
</tr>
<tr>
<td>pulseDetection</td>
<td>26</td>
</tr>
<tr>
<td>pulseIntervals</td>
<td>27</td>
</tr>
<tr>
<td>P_r</td>
<td>27</td>
</tr>
<tr>
<td>radar.range</td>
<td>28</td>
</tr>
<tr>
<td>rainfallDetection</td>
<td>28</td>
</tr>
<tr>
<td>readAudio</td>
<td>29</td>
</tr>
<tr>
<td>referenceIntensity</td>
<td>30</td>
</tr>
<tr>
<td>referencePressure</td>
<td>30</td>
</tr>
<tr>
<td>resonantFrequency</td>
<td>31</td>
</tr>
<tr>
<td>sDuration</td>
<td>31</td>
</tr>
<tr>
<td>sheepFrequencyStats</td>
<td>32</td>
</tr>
<tr>
<td>shimmer</td>
<td>32</td>
</tr>
<tr>
<td>soundSpeed</td>
<td>33</td>
</tr>
<tr>
<td>soundSpeedMedium</td>
<td>33</td>
</tr>
<tr>
<td>soundSpeed_cramer1993</td>
<td>34</td>
</tr>
<tr>
<td>specStats</td>
<td>35</td>
</tr>
<tr>
<td>ste</td>
<td>35</td>
</tr>
</tbody>
</table>
ab_annotations

Get annotations from audioBlast

Description

Search for annotated audio sections on audioBlast (via the ann-o-mate project).

Usage

ab_annotations(...)

Arguments

... Fields and values to filter on. Any field defined by the Ann-o-mate project may be filtered. The most common is likely to be taxon.

Value

A data frame of matching annotations

Examples

## Not run:
ab_annotations(taxon="Gryllotalpa vineae")

## End(Not run)
addSpectra

Description
Search audioBLAST! for recordings with a start time closest to specified date/time which match specified criteria.

Usage
ab_seqss_nearestStart(...)

Arguments
... Fields and values to filter on.

Value
A data frame of matching annotations

Examples
## Not run:
ab_seqss_nearestStart(date="2020-05-15",time="1500")
## End(Not run)

addSpectra

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")

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

Value
A spectrum of s1+s2

Examples

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

## End(Not run)
```

Description

Reads and parses an AudioMoth configuration file.

Usage

`audiomoth_config(filename)`

Arguments

`filename`  Path to the configuration file to read

Value

A data frame of matching annotations

Examples

```r
## Not run:
audiomoth_config("./CONFIG.TXT")

## End(Not run)
```
audio_filesize

Calculated size of raw audio files

Description

Calculates the raw size of audio data at set sample rate, bit depth and duration.

Usage

audio_filesize(
    samp.rate = 44100,
    bit.depth = 16,
    channels = 1,
    duration = 1,
    duration.unit = "seconds",
    output.unit = "bits"
)
**Arguments**

- `samp.rate`: Sample rate
- `bit.depth`: Bit depth
- `channels`: The number of audio channels
- `duration`: Duration of recording
- `duration.unit`: One of seconds, minutes, hours, days
- `output.unit`: "bits" or "bytes"

---

**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**

```r
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

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

**convert2bytes**

Convert bits to bytes

Description

Converts time measurements into seconds

Usage

```r
convert2bytes(S, input = "bits")
```

Arguments

- S: The value to convert
- input: The unit to convert, allowed values are "bits"

Value

The numeric value in seconds
### convert2Celsius

**Convert temperature to Celsius**

**Description**

Converts temperature measurements into Celsius

**Usage**

```r
calculate(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**

```r
calculate(15, input="K")
calculate(15, input="F")
```

---

### convert2dyne_cm2

**Convert pressure to dyne per square centimetre**

**Description**

Converts pressure measurements into dyne per square centimetre

**Usage**

```r
calculate(P, input = "kPa")
```

**Arguments**

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

**Examples**

```r
calculate(1, input="Pa")
calculate(1, input="kPa")
```
**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".

**Examples**
```r
## Not run:
convert2Fahrenheit(15, input = "C")
## End(Not run)
```

**convert2Kelvin**  
*Convert temperature to Kelvin*

**Description**  
Converts temperature measurements into Kelvin

**Usage**  
`convert2Kelvin(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".

**Value**
Numeric value in Kelvin
convert2seconds

convert2Kelvin(15, input="C")
convert2Kelvin(15, input="F")

convert2Pascals  Convert pressure to Pascals

Description
Converts pressure measurements into Pascals

Usage
convert2Pascals(P, input = "kPa")

Arguments
P  The value of the pressure to convert
input  The unit of the pressure to convert, allowed values are "kPa", "dyne_cm2".

Value
The numeric value in Pascals

Examples
convert2Pascals(1000, input="kPa")
convert2Pascals(10, input="dyne_cm2")

convert2seconds  Convert time to seconds

Description
Converts time measurements into seconds

Usage
convert2seconds(T, input = "minutes")

Arguments
T  The time value to convert
input  The unit of time to convert, allowed values are "minutes", "hours", "days", "years". 
cutws

Value

The numeric value in seconds

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,
  remove.offset = TRUE,
  normalise = TRUE
)
```

**Arguments**

- `left`: Data for audio channel
- `samp.rate`: Sampling rate for Wave object
- `bit`: Bit depth of Wave object
- `remove.offset`: If TRUE any DC offset is removed
- `normalise`: IF TRUE the output Wave is normalised using tuneR

**Value**

A mono Wave object.

**Examples**

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

dayPhase

*Phase of day*

**Description**

Given a start time and (optionally) a duration returns the phase of day at a given location. This is primarily used to calculate phase of day information for soundscape recording projects.
dayPhases

Usage

dayPhase(
  time = Sys.time(),
  duration = 2e+05,
  lat = 50.1,
  lon = 1.83,
  tz = "UTC"
)

Arguments

time: A time object representing the start time of a recording
duration: Duration of recording
lat: Latitude of recording device
lon: Longitude of recording device
tz: Time-zone of recording device when recording was made

Value

Data frame of day phases with absolute timestamps and relative times within file

Examples

dayPhase <- function(time=Sys.time(), duration=200000, lat=50.1, lon=1.83, tz="UTC")

Description

Wrapper for suncalc::getSunlightTimes that formats output for this package.

Usage

dayPhases(time, lat, lon, tz)

Arguments

time: A time object representing the start time of a recording
lat: Latitude of recording device
lon: Longitude of recording device
tz: Time-zone of recording device when recording was made
### daysPhases

**Phases of days**

**Usage**

```r
daysPhases(
  date = Sys.Date(),
  period = "year",
  plot = FALSE,
  lat = 50.1,
  lon = 1.83,
  tz = "UTC"
)
```

**Arguments**

- **date**: A time object representing the start time of a recording
- **period**: "month" or "year"
- **plot**: If true plots the data, default FALSE
- **lat**: Latitude of recording device
- **lon**: Longitude of recording device
- **tz**: Time-zone of recording device when recording was made

### 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
**dutyCycle**

**Value**

A cluster object for parallel processing

**Examples**

```r
## Not run:
c1 <- defaultCluster()
stopCluster(c1)
c1 <- defaultCluster(FALSE)
stopCluster(c1)
## End(Not run)
```

---

**dutyCycle**

*Calculate the duty cycle of a wave*

**Description**

Proportion of a wave with signal above the limit

**Usage**

```r
dutyCycle(wave, limit = 0.1, output = "unit", normalise = TRUE)
```

**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".
- `normalise`: If TRUE the Wave is normalised using tuneR

**Value**

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

**Examples**

```r
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
entropyStats(wave)

Arguments
wave A Wave object

Value
A list of spectral entropy types.

Examples
## 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
frequencySound(wl, s = soundSpeedMedium("air"))

Arguments
wl Wavelength
s Speed of sound (defaults to the speed of sound in air)

Value
Frequency of the sound in Hertz
**frequencyStats**

*Examples*

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

---

**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 (uniform, 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.

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**

`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)
```
jitter

Calculate the jitter in a Wave object

Description
Jitter is a measure of the variability of periods in the waveform. Relative jitter is scaled by the jitter in the analysed waveform.

Usage
jitter(wave, method = "absolute")

Arguments

- **wave**
  - A Wave object

- **method**
  - One of "absolute" or "relative"

Value
A vector of zero crossing locations

Examples

```r
## Not run:
jitter(sheep, method="absolute")
jitter(sheep, method="relative")
## 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
labelPadding(t, pad = 0.5, max_t = NULL)

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.
labelReduction

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)
```
naturalFrequency

Calculate the natural frequency

Description

Calculates the natural frequency given the inductance, capacitance and resistance. In the acoustic case the inductance is inertia or mass, the capacitance is elasticity (bulk modulus) and resistance is composed of air resistance and related quantities. All units are SI.

Usage

naturalFrequency(L, C = "default", R)

Arguments

<table>
<thead>
<tr>
<th>L</th>
<th>Inductance</th>
</tr>
</thead>
<tbody>
<tr>
<td>C</td>
<td>Capacitance, by default IUPAC standard pressure.</td>
</tr>
<tr>
<td>R</td>
<td>Resistance</td>
</tr>
</tbody>
</table>

Details

For isothermal compression, the bulk modulus is equal to the pressure. The default value of C therefore is the IUPAC standard pressure.

Examples

f <- naturalFrequency(L=1, C=140, R=12)

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 form the applied function

---

**parseFilename**

**Parse a filename**

**Description**

Attempts to extract meaningful information from a filename.

**Usage**

`parseFilename(string, format = NULL)`

**Arguments**

- **string**: A filename
- **format**: Optionally force a given format - "timestamp"

**Value**

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

**Examples**

`parseFilename("20180605.wav")`
**pd_dietrich2004**  
*Pulse detection using Dietrich (2004)*

**Description**

**Usage**

```r
pd_dietrich2004(
  wave,
  U = 120,
  gamma = 0.05,
  alpha = 1.4,
  scaling = 32,
  V = 480,
  psi = 1
)
```

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

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

---

**pd_simple**  
*Simplified pulse detection using Dietrich (2004)*

**Description**
Detects pulses in a Wave.
pulseDetection

Usage

pd_simple(
    wave,
    U = 120,
    gamma = 0.05,
    alpha = 1.4,
    scaling = 32,
    V = 480,
    psi = 1
)

Arguments

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

pulseDetection Pulse detection

Description

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

Usage

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 side of the mean pulse interval to discard.

**Value**

A list of onset and offset times for pulses.

---

**P_r**

*The radar equation*

**Description**

Calculates the power returned from an echolocation pulse.

**Usage**

```r
P_r(P_t, r, area, G_t = 1, G_r = 1, wl = 1)
```

**Arguments**

- `P_t`: Power transmitted (from sender).
- `r`: Range of the target.
- `area`: Effective cross-sectional area of the target.
- `G_t`: Transmitter gain.
- `G_r`: Receiver gain.
- `wl`: Wavelength (use only with `G_r` and `G_t`).

**Value**

The received power.
Examples

\[ \text{P}_r(12, 20, 0.05) \]
\[ \text{P}_r(12, 20, 0.05, \ G_t=1.2, \ G_r=1.5, \ \text{wl}=0.045) \]

---

**radar.range** *Radar range*

**Description**

Calculates the distance of an object based on the round trip time of an echolocation pulse

**Usage**

\[
\text{radar.range}(t, \ c = \text{soundSpeedMedium}(\text{medium} = \text{"air"}))
\]

**Arguments**

- `t` Time in seconds
- `c` Speed of sound in transmission medium m/s (by default air)

**Value**

Distance to object

**Examples**

\[
\text{radar.range}(2)
\]
\[
\text{radar.range}(2, c=343)
\]
\[
\text{radar.range}(2, c=\text{soundSpeedMedium}(\text{"sea water"}))
\]

---

**fallDetection** *Rainfall detection*

**Description**


**Usage**

\[
\text{fallDetection}(\text{wave, method} = \text{"bedoya2017"}, \ldots)
\]
Arguments

---

**wave**  
A Wave object to detect rainfall in

**method**  
Which rainfall detection method to use ("bedoya2017")

...  
Other arguments to pass to rain detection function

Value

Numeric value from the rainfall detection algorithm chosen.

Examples

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

## End(Not run)
```

---

**readAudio**  
**read an audio file**

Description

This file is used to read an audio file and return a Wave object, it is an abstraction function for various specific audio reading functions. If no existing method can be identified an attempt is made to use the av package to read the audio.

Usage

```r
readAudio(file, mime = "auto", from = 1, to = Inf, units = "samples")
```

Arguments

---

**file**  
File to read

**mime**  
MIME type of file to read, or "auto". Supported types are "audio/x-wav" and "audio/mpeg" (MP3)

**from**  
Start point in file to return

**to**  
End point in file to return

**units**  
One of "samples", "seconds", "minutes", "hours"

Value

A Wave object
**referenceIntensity**     *Reference intensity*

**Description**
Provided the standard reference intensity level.

**Usage**
```r
referenceIntensity(unit = "watt_cm2")
```

**Arguments**
- `unit`     Unit to return, "watt_cm2"

**Examples**
```r
ri <- referenceIntensity()
```

---

**referencePressure**     *Reference pressure*

**Description**
Provided the standard reference pressure level.

**Usage**
```r
referencePressure(unit = "Pa")
```

**Arguments**
- `unit`     Unit to return, "Pa" or "dyne_cm2"

**Examples**
```r
rp <- referencePressure()
rp <- referencePressure(unit="dyne_cm2")
```
resonantFrequency  

*Calculate the resonant frequency*

**Description**

Calculates the resonant frequency given the inductance and capacitance. In the acoustic case the inductance is inertia or mass, the capacitance is elasticity (bulk modulus) and resistance is composed of air resistance and related quantities. All units are SI.

**Usage**

```r
resonantFrequency(L, C = "default")
```

**Arguments**

- **L**: Inductance
- **C**: Capacitance, by default IUPAC standard pressure.

**Details**

For isothermal compression, the bulk modulus is equal to the pressure. The default value of C therefore is the IUPAC standard pressure.

**Examples**

```r
f <- resonantFrequency(L=1)
```

---

sDuration  

*Sample duration*

**Description**

Calculates the time represented by n samples in a Wave.

**Usage**

```r
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
shimmer

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.

shimmer  Calculate the shimmer in a Wave object

Description
Jitter is a measure of the variability of amplitudes within periods in the waveform. Relative shimmer is scaled by the shimmer in the analysed waveform.

Usage
shimmer(wave)

Arguments
wave  A Wave object

Value
A vector of zero crossing locations
soundSpeed

Examples

```r
## Not run:
shimmer(sheep)

## End(Not run)
```

---

**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**

```r
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**

```r
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

<table>
<thead>
<tr>
<th>temp</th>
<th>Temperature</th>
</tr>
</thead>
<tbody>
<tr>
<td>temp.unit</td>
<td>Temperature unit</td>
</tr>
<tr>
<td>pressure</td>
<td>Pressure</td>
</tr>
<tr>
<td>pressure.unit</td>
<td>Pressure unit</td>
</tr>
<tr>
<td>RH</td>
<td>Relative humidity</td>
</tr>
<tr>
<td>MoleFracCO2</td>
<td>Mole fraction of CO2</td>
</tr>
</tbody>
</table>

Value

Numeric value of the speed of sound in m/s

Examples

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**

```
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**

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

**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**

```r
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")
```

**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

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

---

### sweptsine

*Generate a frequency-swept sine wave*

#### Description

Generates a frequency swept sine wave and returns it as a Wave object or vector.

#### Usage

```r
sweptsine(
  f0 = 100,
  f1 = 2500,
  sweep.time = 1,
  A = 1,
  samp.rate = 44100,
  output = "wave",
  ...
)
```

#### Arguments

- `f0`: Start frequency
- `f1`: End frequency
- `sweep.time`: Duration of swept wave
- `A`: Amplitude of wave
- `samp.rate`: Sample rate of swept wave
- `output`: "wave" for a Wave object, or "vector"
- `...`: Additional arguments to pass to data2Wave

#### Value

A swept wave object of the type specified in output.

#### Examples

```r
sweptsine()
```
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)

---

typicalVolume  

Typical volumes

Description

Typical volumes of everyday things.

Usage

typicalVolume(thing = "")

Arguments

ting thing  
Volume of thing, if missing then returns all volumes
Value

Typical volume of thing in dBA, or if no thing parameter a data frame of all volumes

Examples

```r
typicalVolume()
typicalVolume("rocket")
```

### upsample

**Upsample a wave**

#### Description

Used to upsample a Wave object. The upsampled sample rate must be an natural multiple of the current sample rate.

#### Usage

```r
upsample(wave, upsample.rate, method = "basic")
```

#### Arguments

- `wave` Wave object to upsample.
- `upsample.rate` The sample rate to upsample to.
- `method` "basic" for linear, or a function to interpolate NAs in a vector

#### Value

A resampled Wave object

#### Examples

```r
wave <- tuneR::sine(4000, samp.rate=44100)
wave2 <- upsample(wave, 88200)
```
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
code
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
code
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
windowing.functions

Examples

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

## End(Not run)

windowing.functions  List available windowing functions

Description
Lists all available windowing functions.

Usage
windowing.functions()

Examples

## Not run:
windowing.functions()

## End(Not run)

zerocross  Identify zero crossings in a Wave object

Description
Returns a vector of the position (in samples) of zero crossings in a Wave object.

Usage
zerocross(wave)

Arguments

wave  A Wave object

Value
A vector of zero crossing locations

Examples

## Not run:
zerocross(sheep)

## End(Not run)
zeroSpectrum

zeroSpectrum  Zero spectrum

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:
zeroSpectrum(spec)
## End(Not run)
```
Index

* datasets
  sheepFrequencyStats, 32
  STP, 36
* wave
  defaultCluster, 16
  windowing, 40
  windowing.functions, 41
ab_annotations, 3
ab_seqss_nearestStart, 4
addSpectra, 4
audio_filesize, 6
audiomoth_config, 5
audiomoth_wave, 6
autoBandPass, 7
beatComplexity, 8
beatSpectrum, 8
convert2bytes, 9
convert2Celsius, 10
convert2dyne_cm2, 10
convert2Fahrenheit, 11
convert2Kelvin, 11
convert2Pascals, 12
convert2seconds, 12
cutws, 13
data2Wave, 14
dayPhase, 14
dayPhases, 15
daysPhases, 16
defaultCluster, 16
dutyCycle, 17
entropyStats, 18
frequencySound, 18
frequencyStats, 19
generateNoise, 19
gs_transcribe, 20
jitter, 21
labelPadding, 21
labelReduction, 22
naturalFrequency, 23
ntd, 23
P_r, 27
parseFilename, 24
pd_dietrich2004, 25
pd_simple, 25
pulseDetection, 26
pulseIntervals, 27
radar.range, 28
rainfallDetection, 28
readAudio, 29
referenceIntensity, 30
referencePressure, 30
resonantFrequency, 31
sDuration, 31
sheepFrequencyStats, 32
shimmer, 32
soundSpeed, 33
soundSpeed_cramer1993, 34
soundSpeedMedium, 33
specStats, 35
ste, 35
STP, 36
subtractSpectra, 36
sweptsine, 37
tSamples, 38
typicalVolume, 38
upsample, 39
validateIsWave, 40
windowing, 40
windowing.functions, 41
zerocross, 41
zeroSpectrum, 42