Package 'sonicscrewdriver'

May 11, 2024

Title Bioacoustic Analysis and Publication Tools

Version 0.0.7

Description Provides 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 methods, ggplot2, hms, jsonlite, mime, Rdpack, seewave, stringi, suncalc, tuneR

License GPL-3

Encoding UTF-8

LazyData true

RoxygenNote 7.3.1

Language en-GB

Suggests av, covr, devtools, googleCloudStorageR, googleLanguageR, knitr, parallel, pbapply, plotrix, reticulate, rmarkdown, soundecology, spelling, testthat (>= 3.0.0), WaveletComp

VignetteBuilder knitr

URL https://sonicscrewdriver.ebaker.me.uk, https://github.com/edwbaker/SonicScrewdriveR

Config/testthat/edition 3

BugReports https://github.com/edwbaker/SonicScrewdriveR/issues

RdMacros Rdpack

NeedsCompilation no

Author Ed Baker [aut, cre] (<https://orcid.org/0000-0002-5887-9543>), Quentin Geissman [ctb]

Maintainer Ed Baker <ed@ebaker.me.uk>

Repository CRAN

Date/Publication 2024-05-11 20:50:02 UTC

R topics documented:

`*,numeric,PseudoWave-method`

Numeric multiplication by PseudoWave

Description

Numeric multiplication by PseudoWave

Usage

```
## S4 method for signature 'numeric,PseudoWave'
e1 * e2
```

Arguments

`e1`	Input 1
`e2`	Input 2

`*,PseudoWave,numeric-method`

PseudoWave scalar manipulation

Description

PseudoWave scalar manipulation

Usage

```
## S4 method for signature 'PseudoWave,numeric'
e1 * e2
```

Arguments

`e1`	Input 1
`e2`	Input 2

`+,numeric,PseudoWave-method`

Numeric addition by PseudoWave

Description

Numeric addition by PseudoWave

Usage

```
## S4 method for signature 'numeric,PseudoWave'
e1 + e2
```

Arguments

`e1`	Input 1
`e2`	Input 2

`+,PseudoWave,numeric-method`
PseudoWave scalar addition

Description

PseudoWave scalar addition

Usage

```
## S4 method for signature 'PseudoWave,numeric'
e1 + e2
```

Arguments

`e1`	Input 1
`e2`	Input 2

`-,PseudoWave,numeric-method`
PseudoWave scalar subtraction

Description

PseudoWave scalar subtraction

Usage

```
## S4 method for signature 'PseudoWave,numeric'
e1 - e2
```

Arguments

`e1`	Input 1
`e2`	Input 2

/,PseudoWave,numeric-method *PseudoWave scalar division*

Description

PseudoWave scalar division

Usage

```
## S4 method for signature 'PseudoWave,numeric'
e1 / e2
```

Arguments

e1	Input 1
e2	Input 2

ab_diel_traits *Convert text times of day in audioblast traits to numeric values*

Description

This function takes a traits dataset retrieved from audioblast and converts values such as "day" into a numeric time of day based on the date and location.

Usage

```
ab_diel_traits(traits, date, lat, lon, overwrite = FALSE)
```

Arguments

traits	Traits dataset retrieved using audioblast().
date	The date used for conversion for time.
lat	Latitude of location.
lon	Longitude of location.
overwrite	If TRUE then the function will overwrite any existing min/max.

ab_seqss_nearestStart *Nearest start time*

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)
```

addProcess *Add a process to a Tagged Wave or WaveMC object*

Description

This function takes a TaggedWave or TaggedWaveMC object and adds a process to the processing slot. This is used to keep a record of the processes that have been applied to the object.

Usage

```
addProcess(object, process, output = NULL, duration = NULL)

## S4 method for signature 'TaggedWave'
addProcess(object, process, output = NULL, duration = NULL)

## S4 method for signature 'TaggedWaveMC'
addProcess(object, process, output = NULL, duration = NULL)
```

Arguments

object	An object.
process	A description of the process.
output	The output of the process.
duration	The duration of the process in seconds.

Value

The object with the process added.

---

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 = TRUE)
```

Arguments

s1	First spectrum
s2	Second spectrum
coerceNegative	Sets any values below zero to zero in output.

Value

A spectrum of s1+s2

Examples

```
## Not run:
subtractSpectra(spec1, spec2)

## End(Not run)
```

allChannels *Apply a function to all channels of a Wave or WaveMC object*

Description

Some functions (e.g. ffilter from seewave) only operate on a single channel at a time. This function applies the function to each channel and returns a list of analyses.

Usage

```
allChannels(
  w,
  FUN,
  cl = NULL,
  channel.param = "channel",
  output.FUN = NULL,
  ...
)
```

Arguments

w	A Wave or WaveMC object
FUN	Function to apply to the wave.
cl	Optionally a cluster for parallel calculation.
channel.param	Name of the channel parameter to FUN. Can be NULL.
output.FUN	Optional. Function that processes the output of FUN. The "channels_se" function provides standard functionality for the soundecology package.
...	Optional. Additional parameters to pass to FUN.

Value

A list of outputs.

annotation *Create a new Annotation object*

Description

Create a new Annotation object

Usage

```
annotation(
  file = NA_character_,
  metadata = list(),
  start = 0,
  end = Inf,
  low = 0,
  high = Inf,
  source = NA_character_,
  type = NA_character_,
  value = NA_character_
)
```

Arguments

file	File being annotated.
metadata	A list of metadata.
start	Start time of annotation (seconds).
end	End time of annotation (seconds).
low	Low frequency of annotation (Hz).
high	High frequency of annotation (Hz).
source	Source of annotation.
type	Type of annotation.
value	Value of annotation.

Value

An Annotation object.

Annotation-class *A S4 class for annotations*

Description

The Annotation class is used to store annotations on Wave-like objects.

Slots

file File being annotated.

metadata A list for storing metadata.

start Start time of annotation.

end End time of annotation.

low Low frequency of annotation.

`high` High frequency of annotation.

`source` Source of annotation.

`type` Type of annotation.

`value` Value of annotation.

`audioblast` *Get data or analyses from audioBlast*

Description

Search for data or analyses on audioBlast.

Usage

```
audioblast(
  type,
  name,
  endpoint = NULL,
  check = TRUE,
  max_pages = NULL,
  page = 1,
  quiet = FALSE,
  on.issue = stop,
  output = "data.frame",
  ...
)
```

Arguments

`type`	One of data, analysis, standalone.
`name`	Name of data or analysis source.
`endpoint`	Optionally specify endpoint of an audioBlast module.
`check`	Logical. Performs sanity check on input before sending to audioBLAST.
`max_pages`	Maximum number of data pages to return, by default this is set to NULL and returns all pages.
`page`	First page of results to request, defaults to 1.
`quiet`	If true will not print progress. Silence is a virtue.
`on.issue`	Function to call on error or warning. By default `stop` to raise a standard R error. Setting to `warning` will instead a warning.
`output`	By default a `data.frame`. "Annotations" will return a list of `Annotation` objects.
`...`	Fields and values to filter on. Any field defined by audioBLAST.

Value

A data frame

Examples

```
## Not run:
audioblast("data", "recordings", taxon="Gryllotalpa vineae")

## End(Not run)
```

`audioblastDownload` *Download audio files from audioBlast*

Description

Downloads audio files associated with a search using the audioBlast() function.

Usage

```
audioblastDownload(
  d,
  metadata = TRUE,
  skip.existing = TRUE,
  dir = ".",
  quiet = FALSE,
  on.issue = .audioblastIssue
)
```

Arguments

`d`	Data returned from a search using audioBlast().
`metadata`	If true saves the data in d as a csv file.
`skip.existing`	If true will not overwrite existing files.
`dir`	Directory to save files to.
`quiet`	If true will not print progress.
`on.issue`	Function to call on error or warning. By default `stop` to raise a standard R error. Setting to `warning` will instead a warning.

audiomothConfig *Read AudioMoth configuration file*

Description

Reads and parses an AudioMoth configuration file.

Usage

```
audiomothConfig(filename)
```

Arguments

filename	Path to the configuration file to read

Value

A data frame of matching annotations

Examples

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

## End(Not run)
```

audiomothWave *Read AudioMoth metadata from a wave file*

Description

Reads and parses metadata stored in wave files produced by AudioMoth devices.

Usage

```
audiomothWave(filename)
```

Arguments

filename	Path to the wave file to read

Value

A list of extracted parameters

Examples

```
## Not run:
audiomothWave("./FILENAME.WAV")

## End(Not run)
```

---

audio_filesize *Calculated size of raw audio files*

Description

Calculates the raw size of audio date 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

```
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

```
## 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)
```

---

bandpass *Simple bandpass filter*

Description

Creates a band pass WaveFilter between values specified to a Wave object.

Usage

```
bandpass(from, to, ...)
```

Arguments

from	Bottom of bandpass frequency (Hz).
to	Top of bandpass frequency (Hz).
...	Further arguments to pass to ffilter.

Details

This is a simple wrapper function to the seewave ffilter function allowing its use with filterw and pipes.

Value

A WaveFilter object.

Examples

```
## Not run:
nwave <- noise("white", duration=44100, samp.rate=44100)

fwave <- filterWave(nwave, bandpass(from=1000, to=2000))
nwave |> filterWave(bandpass(from=1000, to=2000)) -> fwave

## 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

```
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

```
## 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 rythmicity period expected, in seconds
`max_period`	the maximal rythmicity 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)
```

`birdNetAnalyse` *Analyse sound files using BirdNET-Analyzer*

Description

This function takes a list of sound files and analyses them using the `BirdNET-Analyzer` (Kahl et al. 2021). The function either returns a data frame with the results of the analysis or a list of `Annotation` objects.

Usage

```
birdNetAnalyse(
  files,
  lat = NULL,
  lon = NULL,
  date = NULL,
  output = "Annotation"
)
```

Arguments

`files`	A character vector of file paths.
`lat`	A latitude or vector of latitudes.
`lon`	A longitude or vector of longitudes.
`date`	A `Date` or list of `Date` objects .
`output`	One of "data.frame" or "Annotation".

References

Kahl S, Wood CM, Eibl M, Klinck H (2021). "BirdNET: A deep learning solution for avian diversity monitoring." *Ecological Informatics*, **61**, 101236.

Examples

```
## Not run:
  birdnetAnalyse(files=c("path/to/file1.wav", "path/to/file2.wav"), output="data.frame")

## End(Not run)
```

birdNetInstall *Install the BirdNET environment*

Description

This function installs BirdNET in the `ssd_birdnet` environment using `reticulate`.

Usage

```
birdNetInstall(unattended = FALSE)
```

Arguments

`unattended`	If TRUE then the function will not prompt the user to install the environment in a non-interactive session.

Examples

```
## Not run:
birdnetInstall()
birdNetInstall(unattended=TRUE)

## End(Not run)
```

channels_se *Channels for sound ecology*

Description

Used to process the output of acoustic index functions from the soundecology package when using allChannels.

Usage

```
channels_se(...)
```

Arguments

`...`	Export from a bioacoustic index function from the soundecology package

circularise *Circularise a dataset*

Description

When plotting rings or horizons that are meant to cover the entirety of the time period in a dielPlot() or yearlyPlot() this function append the beginning values to the end to ensure an entire loop is created.

Usage

```
circularise(values)
```

Arguments

values	A vector if values

concat *Concatenate two or more Wave-like objects.*

Description

The concat() method is a more flexible version of the bind() method from tuneR package, that allows specifying more advanced types of concatenation. Setting method to "noClick" will remove any click between Wave objects caused by sudden jumps in amplitude by applying tuneR::prepComb() appropriately with default value of zero (this is only effective for the left channel or stereo or multi-channel recordings).

Usage

```
concat(object, ..., method = "bind")

## S4 method for signature 'Wave'
concat(object, ..., method = "bind")

## S4 method for signature 'WaveMC'
concat(object, ..., method = "bind")

## S4 method for signature 'TaggedWave'
concat(object, ..., method = "bind")

## S4 method for signature 'TaggedWaveMC'
concat(object, ..., method = "bind")
```

Arguments

object	A Wave like object.
...	Wave like objects to concatenate to object.
method	One of "bind", "noClick". Default is "bind".

Value

A concatenated Wave like object, with type of object.

---

convert2bytes *Convert bits to bytes*

Description

Converts time measurements into seconds

Usage

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

Arguments

S	The value to convert
input	The unit to convert, allowed values are "bits", "kB", "MB", "GB"

Value

The numeric value in seconds

---

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")
```

---

convert2degrees *Convert angle to degrees*

Description

Converts angle measurements into degrees

Usage

```
convert2degrees(A, input = "radians")
```

Arguments

A	The angle value to convert
input	The unit of angle to convert, allowed values are "radians".

Value

The numeric value in degrees

---

convert2dyne_cm2 *Convert pressure to dyne per square centimetre*

Description

Converts pressure measurements into dyne per square centimetre

Usage

```
convert2dyne_cm2(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

```
convert2dyne_cm2(1, input="Pa")
convert2dyne_cm2(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

```
## 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

Examples

```
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")
```

convert2radians *Convert angle to radians*

Description

Converts angle measurements into radians

Usage

```
convert2radians(A, input = "degrees")
```

Arguments

A	The angle value to convert
input	The unit of angle to convert, allowed values are "degrees".

Value

The numeric value in radians

convert2seconds *Convert time to seconds*

Description

Converts time measurements into seconds

Usage

```
convert2seconds(T, input = "minutes", origin = "day")
```

Arguments

T	The time value to convert
input	The unit of time to convert, allowed values are "minutes", "hours", "days", "years", "HHMM".
origin	For POSIX whether to return relative to start of day ("day") or Unix epoch ("unix")

Value

The numeric value in seconds

corWaveMC *Correlate channels in a WaveMC object*

Description

Uses the corenv function from seewave to calculate the envelope correlation for timed events between the channels of a WaveMC object

Usage

```
corWaveMC(wave, times, window, temp = 25, cluster = NULL)
```

Arguments

wave	A WaveMC object
times	One or more times of events to correlate
window	Width of the window to correlate in seconds (centred on times)
temp	Air temperature in Celsius
cluster	A cluster for parallel execution

Value

List of corenv lists for events, and a list of the time differences between channels

cutws *Cut wave by samples*

Description

Extract a section of a Wave object based on sample positions. This function will automatically detect if a Wave object is stereo.

Usage

```
cutws(wave, from = 1, to = Inf, 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

```
data2Wave(
  left,
  samp.rate = 44100,
  bit = 16,
  unit = NULL,
  remove.offset = TRUE,
  normalise = TRUE
)
```

Arguments

`left`	Data for mono audio channel
`samp.rate`	Sampling rate for Wave object
`bit`	Bit depth of Wave object
`unit`	See tuneR::normalize. If NULL this is handled automatically.
`remove.offset`	If TRUE any DC offset is removed
`normalise`	IF TRUE the output Wave is normalised to -1:1

Value

A mono Wave object.

Examples

```
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.

Usage

```
dayPhase(
  time = Sys.time(),
  duration = 40000,
  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

dayPhases *Phases of day*

Description

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

Usage

```
dayPhases(time = as.Date(Sys.time()), lat = 50.1, lon = 1.83, tz = "UTC")
```

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*

Description

Phases of days

Usage

```
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 is not be available on Windows. Hence, the fork parameter has no effect on Windows.

Usage

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

Arguments

fork	If TRUE uses forking to create the cluster (Unix like systems only)

Value

A cluster object for parallel processing

Examples

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

## End(Not run)
```

dielFraction *Calculate the fraction of a day given by a value*

Description

Given an object that can be coerced to POSIXlt or is in a supported string format, return the fraction of a day represented by the object.

Usage

```
dielFraction(t, input = "POSIX", unit = "radians")
```

Arguments

t	Object to be converted to a fraction
input	One of POSIX (default) or HHMM
unit	If set to radians outputs a position around a circle. If set to fraction outputs the raw fraction.

`dielHistogram` *Diel Histogram*

Description

Draws a histogram on a dielPlot() using pre-defined bins related to time of day.

Usage

```
dielHistogram(
  times,
  by = "hour",
  col = "grey",
  maxval = NA,
  presence.only = FALSE,
  limits = c(1, 2)
)
```

Arguments

times	A vector of times that can be pocessed by dielFraction().
by	Controls the size of histogram bins, one of "hour", "15minute", "30minute".
col	Colour of the plot.
maxval	By default scales histogram within limits, specifying a maximum value here allows comparison between plots.
presence.only	Only show presence/absence not values.
limits	Limits of the plotting (see dielPlot()).

Value

A data frame of start and end points of bins.

dielLabels *Generate labels for a diel plot*

Description

Generates labels for a dielPlot() in 12- or 24-hour format. Labels are generated at three hourly intervals.

Usage

```
dielLabels(format = "clock24")
```

Arguments

format	One of clock24 (default) or clock12

Examples

```
dielLabels()
dielLabels("clock12")
```

dielPlot *Create a diel plot*

Description

A diel plot shows the times of night, twilight and the maximum altitude of the sun for a given date.

Usage

```
dielPlot(
  date,
  lat,
  lon,
  limits = c(0, 2),
  plot = NULL,
  rot = tzRot(0),
  method = "plotrix",
  legend = F
)
```

Arguments

date	Date to plot.
lat	Numeric latitude.
lon	Numeric longitude.
limits	Plotting limits of the daylight regions, default to c(1,2)
plot	Character vector of components to plot
rot	Either "Solar Noon" or an offset calculated by tz
method	Plotting library to use
legend	Whether to show a legend

---

dielPositions *Generate positions of labels for a diel plot*

Description

Generates positions for three-hourly labels of a dielPlot() in radians.

Usage

```
dielPositions(format = "3hourly")
```

Arguments

format	One of "3hours" (default), "hours", or "minutes"

Examples

```
dielPositions()
dielPositions("hours")
dielPositions("minutes")
```

---

dielRings *Plot rings on a diel plot*

Description

Plot rings on a diel plot.

Usage

```
dielRings(
  names,
  starts,
  ends,
  cols = "grey",
  format = "HHMM",
  limits = c(1, 2),
  legend = T
)
```

Arguments

names	Labels for the rings
starts	Start times for rings in HHMM string format
ends	End times for rings in HHMM string format
cols	Colours of the rings
format	Defaults to HHMM
limits	Region of a dielPlot() to plot rings. Defaults to c(1,2)
legend	Boolean. Whether to plot a legend.

---

dolbear *Dolbear's law*

Description

Calculates either chirps per minute based on temperature or vice versa using Dolbear's law (or equivalent laws for other species)

Usage

```
dolbear(n = NULL, t = NULL, species = "Oecanthus fultoni")
```

Arguments

n	Chirps per minute
t	Temperature in Celsius
species	Species to use (by default Oecanthus fultoni)

Value

Missing value of n or t

Examples

```
dolbear(n=6)
dolbear(t=25)
```

`dutyCycle` *Calculate the duty cycle of a wave*

Description

Proportion of a wave with signal above the limit

Usage

```
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% if percentage output).

Examples

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

`emptyDiel` *Create an empty diel plot*

Description

Create a diel plot with labels but without sun altitude or times of day plotted.

Usage

```
emptyDiel(method = "plotrix", rot = pi)
```

Arguments

`method`	Plotting package to use
`rot`	Rotation of the origin (defaults to pi)

emptyYearly *Create an empty yearly plot*

Description

Create a yearly plot with labels but without sun or night duration plotted.

Usage

```
emptyYearly(year = 2022, method = "plotix", rot = pi)
```

Arguments

year	Year to plot (allows for leap years)
method	Plotting package to use
rot	Rotation of the origin (defaults to pi)

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)
```

filterWave *Apply a WaveFilter object to a Wave object*

Description

A WaveFilter object is an object containing information necessary for the filterw function to apply the filter to a Wave object. This is designed to allow a pipe operator (either magrittr or base R) to be used to apply filters to a Wave in a pipeline.

Usage

```
filterWave(w, filt, cl = NULL)
```

Arguments

w	A Wave object.
filt	Wave object with the selected filter applied.
cl	Optional. If a cluster is specified, the filter will be applied in parallel.

Details

Supported filters include those from the seewave package.

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 = soundSpeed(medium = "air"))
```

Arguments

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

Value

Frequency of the sound in Hertz

Examples

```
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

```
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 Wave like object*

Description

Adding noise to a Wave like object allows for testing of the robustness of automated identification algorithms to noise.

Usage

```
generateNoise(
  wave,
  noise = c("white"),
  noise.add = FALSE,
  noise.ratio = 0.5,
  noise.ref = "file",
  output = "list"
)
```

Arguments

wave	Object to add noise to (Wave, WaveMC, or Tagged versions), or a list of such objects.
noise	Vector of noise to add (unif, gaussian, white, pink, power, red)
noise.add	If TRUE all noise sources are added to wave. If FALSE separate outputs are created for each noise source.
noise.ratio	Ratio of maximum noise amplitude to the maximum amplitude in wave.
noise.ref	Reference maximum for noise.ratio. If "max" then the maximum amplitude, if "file" then the maximum amplitude of wave.
output	TODO: Is this implemented?

Value

A list of Wave objects with the required noise added.

---

generateTimeMask *Generate time masked Wave-like objects*

Description

Given a Wave-like object (or a list of Wave-like objects), generate new Wave-like objects with time masking.

Usage

```
generateTimeMask(wave, method = "squarewave", dutyCycle = 0.95, n.periods = 10)
```

Arguments

wave	A Wave-like object (or a list of Wave-like objects).
method	The method to use for time masking (one of "squarewave", "random).
dutyCycle	The duty cycle of the output. A value of 0.95 means that 5% of the time is masked.
n.periods	The number of waves to generate in the squarewave method.

generateTimeShift *Generated time-shifted versions of a Wave-like object*

Description

Given a Wave-like object (or list of Wave-like objects), this function generates time-shifted versions of the object. The time-shifted versions are generated by adding a constant amount of time to the start or end of the object. This is achieved by either inserting silence and truncating the object to the original length, or by rotating the audio within the object.

Usage

```
generateTimeShift(
  wave,
  type = "silent",
  amount = c(1, 2),
  where = "start",
  output = "list"
)
```

Arguments

wave	A Wave-like object or list of Wave-like objects.
type	The type of time-shift to apply. Either "silent" or "rotate".
amount	Vector of amount of time to shift by (seconds).
where	Where to insert silence if type is "silent".
output	Return a list.

Value

A Wave-like object or list of Wave-like objects.

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

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

## End(Not run)
```

---

humanBytes *Converts bytes in human readable form*

Description

Given an input of bytes calculates the result in a sensible output unit (e.g. MB, GB, PB).

Usage

```
humanBytes(S)
```

Arguments

S	Number of bytes

Value

String in human readable format

humanTime *Converts time to human readable form*

Description

Given an input of bytes calculates the result in a sensible output unit (e.g. minutes, hours).

Usage

```
humanTime(S, unit = "seconds")
```

Arguments

S	Time to convert in unit
unit	The unit of time to convert

Value

String in human readable format

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

```
## 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.

Value

A modified Google Speech API transcript object

Examples

```
## 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

```
labelReduction(t)
```

Arguments

t	Transcript from Google Speech API

Value

A list containing start and end times of speech containing regions

Examples

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

## End(Not run)
```

`map2RGB` *Map three vectors to RGB*

Description

Maps three vectors of equal length to RGB for use in false-colour index spectrograms

Usage

```
map2RGB(red, green, blue)
```

Arguments

`red`	The red channel vector
`green`	The green channel vector
`blue`	The blue channel vector

Value

A vector of RGB values

`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

`L`	Inductance
`C`	Capacitance, by default IUPAC standard pressure.
`R`	Resistance

Details

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

Examples

```
naturalFrequency(L=20,R=0.5)
naturalFrequency(L=20,C=1/4,R=0.5)
```

`normalise` *Normalise a Wave object*

Description

Similar to normalize() from the tuneR package but automatically identifies the unit parameter.

Usage

```
normalise(wave, unit = NULL, ...)
```

Arguments

`wave`	Wave or WaveMC object
`unit`	If not null behaves as in normalize() from tuneR, if null the unit is automatically identified.
`...`	Additional arguments passed to normalize() from tuneR

Value

Normalised Wave or WaveMC object

`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, typically the date and time a recording started.

Usage

```
parseFilename(file, format = NULL, timezone = NULL)
```

Arguments

file	A filename (or list of filenames).
format	Optionally force a given format (see Details). If NULL (default) an attempt is made to automatically detect the format for each file. If "match" and a list of filenames is given then an attempt will be made to find a format that matches all files. This may give incorrect results if the filename is ambiguous (see Details).
timezone	Optionally set a timezone.

Details

Determining the format:

It is sometimes impossible to accurately determine the format of a filename, e.g. when an eight-digit 'AudioMoth HEX' only contains numbers it could be confused with a YYYYMMDD format. If a list of filenames is given and the "match" format is specified then an effort will be made to determine the most likely format that applies to all filenames.

Supported formats:

- **AudioMoth** - The newer format for AudioMoth devices consists of a standard YYYYMMDD_HHMMSS.wav format. Specifying 'AudioMoth' forces a call to the `audiomoth()` function from the `seewave` package (Sueur et al. 2008).
- **AudioMoth HEX** - Older format for AudioMoth devices consisting of eight hexadecimal characters. Conversion is handled by a call to `seewave::audiomoth()`.
- **timestamp** - A standard date-time format. Uses the R standard origin of 1970-01-01 00:00:00 UTC.
- **Wildlife Acoustics SM2** - Can also be used for Wildlife Acoustics SM4 devices. Conversion is handled by a call to `seewave::songmeter()`.
- **Wildlife Acoustics SM3** - Conversion is handled by a call to `seewave::songmeter()`.
- **YYYYMMDD_HHMMSS** - A standard date-time format.

Value

A list of file, type of match, datetime.

It is possible to determine additional properties from some files, these will be added to the list.

References

Sueur J, Aubin T, Simonis C (2008). "Seewave, a free modular tool for sound analysis and synthesis." *Bioacoustics*, **18**(2), 213–226.

Examples

```
parseFilename("5E90A4D4.wav")
```

`pd_dietrich2004` *Pulse detection using Dietrich (2004)*

Description

Detects pulses in a Wave using the method described in Dietrich et al (2004) doi:10.1016/j.patcog.2004.04.004.

Usage

```
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.

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

pseudoWave *Create a PseudoWave object*

Description

This function is used to create a PseudoWave object that can be used to generate a Wave object when operated on.

Usage

```
pseudoWave(
  type = NA_character_,
  subtype = NA_character_,
  scale = 1,
  offset = 0,
  seed = 1,
  params = list()
)
```

Arguments

type	Type of PseudoWave (e.g. "noise", "sine")
subtype	Subtype of PseudoWave (e.g. "white" if type is "noise")
scale	The Wave channels are multiplied by this value
offset	This value is added to the Wave channels
seed	Random seed for reproducible output. NA for no
params	List of additional parameters to pass to generating function

Value

A PseudoWave object.

Examples

```
pw <- pseudoWave("noise", "white")

pw <- pseudoWave("sine", params=list("f0"=440))
```

PseudoWave-class *An S4 class to represent a PseudoWave object that is converted to a Wave object when operated on.*

Description

An S4 class to represent a PseudoWave object that is converted to a Wave object when operated on.

Slots

`type` Type of PseudoWave (e.g. "noise")

`subtype` Subtype of PseudoWave (e.g. "white" if type is "noise")

`scale` The Wave channels are multiplied by this value

`offset` This value is added to the Wave channels

`seed` Random seed for reproducible output, NA for no seed

`scale` Logical. Whether to use the random seed value

`params` List of additional parameters to pass to generating function

pulse *Generate a single pulse*

Description

Generate a single pulse, either a Dirac pulse (Dirac delta) or a square pulse.

Usage

```
pulse(
  type = "dirac",
  leading = 22050,
  pulse.length = 1,
  duration = samp.rate,
  samp.rate = 44100,
  bit = 1,
  pcm = FALSE,
  stereo = FALSE,
  output = "Wave",
  invert = FALSE
)
```

Arguments

type	Either "dirac" or "square".
leading	The number of samples before the pulse.
pulse.length	The number of samples in the pulse (for "square").
duration	The total number of samples generated.
samp.rate	The sample rate.
bit	The bit depth.
pcm	Whether Wave generated is PCM (see tuneR).
stereo	Whether Wave generated is stereo.
output	The output format ("Wave").
invert	Whether to invert the pulse.

Value

Specified by output.

---

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

```
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

radarPower *The radar equation*

Description

Calculates the power returned from an echolocation pulse

Usage

```
radarPower(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

```
radarPower(12, 20, 0.05)
radarPower(12, 20, 0.05, G_t=1.2, G_r=1.5, wl=0.045)
```

`radarRange` *Radar range*

Description

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

Usage

```
radarRange(t, c = soundSpeed(medium = "air"))
```

Arguments

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

Value

Distance to object

Examples

```
radarRange(2)
radarRange(2, c=343)
radarRange(2, c=soundSpeed(medium = "sea water"))
```

`radialPolygon` *Plot a radial polygon*

Description

Used to plot sectors, annuli and horizons on a dielPlot() or yearlyPlot(). The polygon has an inner and outer horizon - which can be set to a fixed radius or a vector.

Usage

```
radialPolygon(
  angle1,
  angle2,
  radius1,
  radius2,
  col = "grey",
  border = NA,
  rot = -pi,
  angleinc = 0.01,
  reverse = TRUE,
  ...
)
```

Arguments

angle1	Angles for the inner line
angle2	Angles for the outer line
radius1	Radii for the inner line
radius2	Radii for the outer line
col	Colour of the polygon
border	Border colour (see polygon() for details)
rot	Rotation of the plot, defaults to pi to match dielPlot() and yearlyPlot()
angleinc	The angular increment in radians for calculating circular lines
reverse	If FALSE plots in an anti-clockwise direction
...	Other parameters passed to polygon()

---

rainfallDetection *Rainfall detection*

Description

Detects rainfall in a Wave. An uncalibrated version of Bedoya et al (2017) doi:10.1016/j.ecolind.2016.12.018 is available in this package. The hardRain package can also be accessed via this wrapper.

Usage

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

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

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

## End(Not run)
```

`readAudacityLabels` *Read an Audacity label file*

Description

Reads an Audacity label file and returns either a list of `Annotation` objects or a data frame.

Usage

```
readAudacityLabels(file, output = "annotations")
```

Arguments

`file`	Path to the Audacity label file.
`output`	One of "annotations" or "data.frame".

`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

```
readAudio(file, mime = "auto", from = 0, to = Inf, units = "seconds")
```

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". Default is "seconds".

Value

A Wave object

---

readBirdNet *Read output files from BirdNet Analyser*

Description

Reads a single file, or directory of files, output by BirdNet Analyser.

Usage

```
readBirdNet(file, filename_parsing = "none")
```

Arguments

file	Filename or directory
filename_parsing	Allows for filename parsing, accepted values are one of none, audiomoth, timestamp.

Value

A data frame.

readRespeaker6 *Read a file from Seeed Studio Respeaker 6 mic array*

Description

The Seeed Studio Respeaker-6 when used as described in the documentation saves an eight channel audio file with channels 7 and 8 not containing input audio. This function reads such a file and saves it as a six channel file.

Usage

```
readRespeaker6(filename, from = 1, to = Inf, units = "samples", header = FALSE)
```

Arguments

filename	file to read.
from	Where to start reading the wave in units.
to	Where to stop reading the wave in units.
units	Units in which from and to is given, the default is "samples", but can be set to time intervals such as "seconds".
header	If TRUE, just header information of the Wave file are returned, otherwise (the default) the whole Wave object.

Value

A WaveMC object.

referenceIntensity *Reference intensity*

Description

Provides the standard reference intensity level.

Usage

```
referenceIntensity(unit = "watt_cm2")
```

Arguments

unit	Unit to return, "watt_cm2"

Examples

```
ri <- referenceIntensity()
```

referencePressure *Reference pressure*

Description

Provides the standard reference pressure level.

Usage

```
referencePressure(unit = "Pa")
```

Arguments

unit	Unit to return, "Pa" or "dyne_cm2"

Examples

```
rp <- referencePressure()
rp <- referencePressure(unit="dyne_cm2")
```

region *Specify a region with a file to analyse*

Description

Specifies a time-bounded region to analyse.

Usage

```
region(unit, from = 0, to = Inf)
```

Arguments

unit	Unit of time (one of samples, seconds, minutes, hours)
from	Start time
to	End time

Value

A TimeRegion object.

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

```
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

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

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

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

Examples

```
## 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

```
soundSpeed(
  medium = NULL,
  method = NULL,
  wl = NULL,
  f = NULL,
  bulkModulus = NULL,
  density = NULL,
  ...
)
```

Arguments

medium	Propagation medium (e.g. "air"), or "all" to return a list of all available media.
method	Use a specific method to calculate the speed of sound (see Details).
wl	Wavelength
f	Frequency
bulkModulus	Bulk modulus
density	Density
...	Additional parameters passed to the method.

Details

The speed of sound can be calculated using the following methods:

- **cramer** Uses the method described in Cramer (1993). Additional parameters are:
  - temp Temperature
  - temp.unit Temperature unit
  - pressure Pressure
  - pressure.unit Pressure unit

  - RH Relative humidity
  - MoleFracCO2 Mole fraction of CO2
- **seewave** Delegates the calculation of the speed of sound in air to the package `seewave` (Sueur et al. 2008). This calculation is . performed as speed $= 331.4 + 0.6 \times$ temp. Additional parameters are:
  - temp Temperature

References

Cramer O (1993). "The variation of the specific heat ratio and the speed of sound in air with temperature, pressure, humidity, and CO2 concentration." *The Journal of the Acoustical Society of America*, **93**(5), 2510-2516. ISSN 0001-4966, doi:10.1121/1.405827.

Sueur J, Aubin T, Simonis C (2008). "Seewave, a free modular tool for sound analysis and synthesis." *Bioacoustics*, **18**(2), 213–226.

Examples

```
soundSpeed(medium="air")
soundSpeed(medium="sea water")

soundSpeed(method="cramer", temp=14, pressure=3, RH=10)
soundSpeed(method="cramer", temp=14, temp.unit="C", pressure=3, pressure.unit="kPa", RH=10)

t <- 1:30
s <- lapply(t, \(x){soundSpeed(method="cramer", temp=x)})
```

`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
method	Which method used to calculate the short term energy, by default "dietrich2004" to use (Dietrich et al. 2004).
...	Other arguments to pass to ste method.

Value

A vector of short term energy values

References

Dietrich C, Palm G, Riede K, Schwenker F (2004). "Classification of bioacoustic time series based on the combination of global and local decisions." *Pattern Recognition*, **37**(12), 2293–2305.

Examples

```
## 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

```
subtractSpectra(s1, s2, coerceNegative = TRUE)
```

Arguments

`s1`	First spectrum
`s2`	Second spectrum
`coerceNegative`	Sets any values below zero to zero in output.

Value

A spectrum of s1 - s2

Examples

```
## Not run:
subtractSpectra(spec1, spec2)
subtractSpectra(spec1, spec2, coerceNegative=TRUE)

## End(Not run)
```

`sweptsine` *Generate a sine sweep*

Description

Generates a frequency swept sine wave (either linear or logarithmic) and returns it as a Wave object or vector.

Usage

```
sweptsine(
  f0 = 100,
  f1 = 2500,
  mode = "linear",
  sweep.time = 1,
  time.unit = "seconds",
  samp.rate = 44100,
  output = "wave",
  ...
)
```

Arguments

f0	Start frequency
f1	End frequency
mode	One of "linear", "log"
sweep.time	Duration of swept wave
time.unit	One of "seconds", "samples"
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

```
#Generate a swept sine wave between 0Hz and 10kHz.
w <- sweptsine(0, 10e3)

#Generate a swept sine wave between 0Hz and 10kHz and normalise it.
w <- normalise(sweptsine(0, 10e3))

#Generate a stereo swept sine wave between 100Hz and 1KHz.
w <- tuneR::stereo(sweptsine(100, 1e3))

#Generate an exponentially swept sine wave between 100Hz and 1KHz.
w <- sweptsine(100, 1e3, mode="log")
```

TaggedWave-class *A S4 class for tagged waves*

Description

The TaggedWave class extended the Wave class from the tuneR package so that it can include extended metadata and the results of analyses.

Slots

`metadata` A list for storing metadata.

`analyses` A list for storing analyses.

TaggedWaveMC-class *A S4 class for tagged multi-channel waves*

Description

The TaggedWaveMC class extended the WaveMC class from the tuneR package so that it can include extended metadata and the results of analyses.

Slots

`metadata` A list for storing metdata.

`analyses` A list for storing analyses.

tagWave *Tag a Wave or WaveMC object*

Description

This function takes a `Wave`/`WaveMC` object (or a list of such objects) and returns a corresponding tagged version (`TaggedWave` or `TaggedWaveMC`).

Usage

```
tagWave(w, origin = "user")
```

Arguments

`w`	A `Wave` or `WaveMC` object (or list of such objects).
`origin`	The origin of the object (default "user").

Value

A `TaggedWave` or `TaggedWaveMC` object (or list of such objects).

TimeRegion-class *An S4 class to represent a TimeRegion within a Wave object.*

Description

An S4 class to represent a TimeRegion within a Wave object.

Slots

`from` Start position

`to` End position

`unit` Time unit (one of seconds, minutes, hours)

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 = NA_character_)
```

Arguments

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

```
typicalVolume()
typicalVolume("rocket launch")
```

tzRot *Converts a timezone offset into a rotation*

Description

Given a timezone offset in hours returns a rotation in radians to apply to values for a diel plot.

Usage

```
tzRot(tz, init = pi)
```

Arguments

tz	Timezone numeric
init	Initial rotation. Defaults to pi.

untagWave *Untag a TaggedWave or TaggedWaveMC object*

Description

This function takes a TaggedWave/TaggedWaveMC object (or a list of such objects) and returns a corresponding Wave/WaveMC object (or list of such objects).

Usage

```
untagWave(w)
```

Arguments

w	A TaggedWave or TaggedWaveMC object (or list of such objects).

Value

A Wave or WaveMC object.

Examples

```
## Not run:
w <- noise("white")
tw <- tagWave(w)
w2 <- untagWave(tw)

## End(Not run)
```

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

```
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

```
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

```
validateIsWave(wave)
```

Arguments

`wave`	Object to test

`WaveFilter-class` *WaveFilter object for audio filters*

Description

A `WaveFilter` object is an object containing information necessary for the `filterWave()` function to apply the filter to a `Wave` or `TaggedWave` object. This is designed to allow a pipe operator (either magrittr or base R) to be used to apply filters to a Wave in a pipeline. If used with a `TaggedWave` object the function adds information to the `processing` slot documenting its action.

Slots

`description` Description of the filter.

`func` Name of function.

`params` List of additional parameters to pass to the function.

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 multi-core processing. It will also show a progress bar if the 'pbapply' package is installed.

Usage

```
windowing(
  wave,
  window.length = 1000,
  FUN,
  window.overlap = 0,
  bind.wave = FALSE,
  complete.windows = TRUE,
  cluster = NULL,
  ...
)
```

Arguments

wave	A Wave object or filename. Using filenames may save loading an entire large file into memory.
window.length	The length of the analysis window (in samples).
FUN	FUN to be applied to windows.
window.overlap	The overlap between successive windows (in samples), a negative value will result in a gap between windows.
bind.wave	If TRUE and FUN returns wave objects, then these are combined into a single object
complete.windows	If TRUE (default) the final window will not be processed unless it has a length equal to window.length.
cluster	A cluster form the 'parallel' package for multi-core computation.
...	Additional parameters to FUN

Examples

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

## End(Not run)
```

writeAudacityLabels *Write an Audacity label file*

Description

Writes a list of `Annotation` objects to an Audacity label file.

Internally this uses the `write.audacity()` function from the `seewave` package (Sueur et al. 2008).

Usage

```
writeAudacityLabels(annotations, file)
```

Arguments

`annotations`	A list of `Annotation` objects.
`file`	Path to the Audacity label file.

References

Sueur J, Aubin T, Simonis C (2008). "Seewave, a free modular tool for sound analysis and synthesis." *Bioacoustics*, **18**(2), 213–226.

yearlyFraction *Calculate the fraction of a year given by a value*

Description

Given an object that can be coerced to POSIXlt, return the fraction of a year represented by the object.

Usage

```
yearlyFraction(t, year = 2022, input = "POSIX", unit = "radians")
```

Arguments

`t`	Object to be converted to a fraction
`year`	Year to calculate fractions of (allows for leap years)
`input`	One of POSIXlt (default)
`unit`	If set to radians outputs a position around a circle. If set to fraction outputs the raw fraction.

yearlyLabels *Generate labels for a yearly plot*

Description

Generates monthly labels for a yearlyPlot()..

Usage

```
yearlyLabels()
```

yearlyPlot *Create a yearly plot*

Description

ToDO.....

Usage

```
yearlyPlot(
  year = 2022,
  lat,
  lon,
  limits = c(0, 2),
  plot = NULL,
  method = "plotrix",
  legend = F
)
```

Arguments

year	Year to plot (allows for leap years).
lat	Numeric latitude.
lon	Numeric longitude.
limits	Plotting limits of the daylight regions, default to c(1,2)
plot	Character vector of components to plot
method	Plotting library to use
legend	Whether to show a legend

yearlyPositions *Generate positions of labels for a yearly plot*

Description

Generates positions for monthly labels of a dielPlot() in radians. The positions can either be for the start of the month, or middle of the month.

Usage

```
yearlyPositions(year = 2022, format = "months")
```

Arguments

year	Year to calculate
format	One of months, mid-months, days

Details

The function allows for leap years if the year parameter is provided.

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 *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

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

## End(Not run)
```

[,Wave,TimeRegion-method *Allow subsetting a Wave object with a TimeRegion*

Description

Allow subsetting a Wave object with a TimeRegion

Usage

```
## S4 method for signature 'Wave,TimeRegion'
x[i]
```

Arguments

x	Wave Object
i	TimeRegion object

Index