Package 'PAMmisc'

February 24, 2025

Title Miscellaneous Functions for Passive Acoustic Analysis

Version 1.12.4

Description A collection of miscellaneous functions for passive acoustics.
Much of the content here is adapted to R from code written by other people.
If you have any ideas of functions to add, please contact Taiki Sakai.

License GNU General Public License

Encoding UTF-8

LazyData true

RoxygenNote 7.3.1

Imports ggplot2, tuneR, seewave, dplyr, RcppRoll, PamBinaries,
RSQLite, lubridate, rerddap, ncdf4, httr, purrr, xml2, methods,
geosphere, tcltk, scales, suncalc, rjson, fftw, signal

Suggests testthat

Depends R (>= 3.50)

NeedsCompilation yes

Author Taiki Sakai [aut, cre],
Jay Barlow [ctb],
Julie Oswald [ctb],
Val Schmidt [ctb]

Maintainer Taiki Sakai <taiki.sakai@noaa.gov>

Repository CRAN

Date/Publication 2025-02-24 19:10:02 UTC

Contents

---

addPgAnno *Add Spectrogram Annotations to Pamguard Database*

Description

Add new annotations to an existing Pamguard Spectrogram Annotations table

Usage

```
addPgAnno(
  db,
  anno,
  tableName = NULL,
  channel = 1,
  source = c("manual", "aplose", "pammisc", "annomate", "raven"),
  format = c("%m/%d/%Y %H:%M:%OS", "%m-%d-%Y %H:%M:%OS",
```

```
    "%Y/%m/%d %H:%M:%OS", "%Y-%m-%d %H:%M:%OS"),
  tz = "UTC"
)
```

Arguments

db	database file to add annotations to
anno	annotations to add, must contain columns UTC, Duration (seconds), f1 (min freq Hz), and f2 (max freq Hz). Any other columns matching columns in the database will also be added
tableName	name of the annotation table in the database
channel	channel to display the annotations on
source	annotation source. If 'manual', columns UTC, DUration, f1, and f2 must be present. Other options will attempt to automate conversion to these column names from specific output sources
format	date format, default will try two variations of MDY HMS and YMD HMS
tz	timezone of provided date

Value

Returns a dataframe of the rows added to the database

Author(s)

Taiki Sakai <taiki.sakai@noaa.gov>

Examples

```
## Not run:
myDb <- 'PamguardDatabase.sqlite3'
myAnno <- data.frame(UTC = '2021/10/23 12:10:10', Duration = .563, f1=2300, f2=3600)
addPgAnno(myDb, myAnno, tableName='Spectrogram_Annotation', source='manual')

## End(Not run)
```

---

addPgEvent *Add Pamguard Event to Database*

Description

Add a new event to an existing Pamguard database in the "OfflineEvents" table. If the specified eventType does not exist in the database, it will be added to the "Lookup" table.

Usage

```
addPgEvent(
  db,
  UIDs = NULL,
  binary,
  eventType,
  comment = NA,
  tableName = NULL,
  start = NULL,
  end = NULL,
  type = c("click", "dg")
)
```

Arguments

db	database file to add an event to
UIDs	vector of the UIDs of the individual detections to add to the event
binary	binary file(s) containing the detections from UIDs
eventType	the name of the event type to add. If this is not already present in the database, it will be added to the "Lookup" table
comment	(optional) a comment for the event
tableName	(optional) specify the name of the Click Detector that generated the event table you want to add to. This only needs to be specified if you have more than one click detector, it defaults to the first "NAME_OfflineEvents" table in the database.
start	(optional) start time of event. Mandatory if no detections are added
end	(optional) end time of event. Mandatory if no detections are added
type	type of event data to add, either 'click' to add event data using the Click Detector module, or 'dg' to add event data using the Detection Grouper module

Value

Adds to the database db, invisibly returns TRUE if successful

Author(s)

Taiki Sakai <taiki.sakai@noaa.gov>

Examples

```
## Not run:
myDb <- 'PamguardDatabase.sqlite3'
myBinaries <- c('./Binaries/Bin1.pgdf', './Binaries/Bin2.pgdf')
addUIDs <- c(10000001, 10000002, 20000007, 20000008)
addPgEvent(db = myDb, UIDs = addUIDs, binary = myBinaries, eventType = 'MyNewEvent')

## End(Not run)
```

addPgGps *Add GPS to a Pamguard Database*

Description

Add GPS data to an existing Pamguard database

Usage

```
addPgGps(
  db,
  gps,
  source = c("csv", "SPOTcsv", "SPOTgpx"),
  format = c("%m/%d/%Y %H:%M:%S", "%m-%d-%Y %H:%M:%S",
    "%Y/%m/%d %H:%M:%S", "%Y-%m-%d %H:%M:%S"),
  tz = "UTC"
)
```

Arguments

db	database file to add gps data to
gps	data.frame of gps data or a character of the file name to be read. If a data.frame or non-SPOT csv file, needs columns UTC, Latitude, and Longitude. If multiple separate tracks are present in the same dataset, this should be marked with a column labeled Name
source	one of SPOTcsv, SPOTgpx, or csv. Describes the source of the GPS data, not needed if gps is a data.frame
format	date format for converting to POSIXct, only needed for source='csv'. See strptime
tz	timezone of gps source being added, will be converted to UTC

Value

Adds to the database db, invisibly returns the Name of the GPS track if successful (NA if not named)

Author(s)

Taiki Sakai <taiki.sakai@noaa.gov>

Examples

```
## Not run:
# not run because example files don't exist
myDb <- 'PamguardDatabase.sqlite3'
# adding from a .gpx file downloaded from SPOT
spotGpx <- 'SpotGPX.gpx'
addPgGps(myDb, spotGpx, source='SPOTgpx')
```

```
# adding from a csv file with a Y-M-D H:M date format
gpsCsv <- 'GPS.csv'
addPgGps(myDb, gpsCsv, source='csv', format='%Y-%m-%d %H:%M')

## End(Not run)
```

`browseEdinfo` *Browse a List of Environmental Datasets*

Description

This function browses the list of selected environmental datasets that are recommended as a starting point, and prompts the user to select one to use, returning an edinfo object. Also allows user to filter by variable name, matching will be attempted using regex

Usage

```
browseEdinfo(var = NULL)
```

Arguments

`var`	the name or partial name of a variable to filter the available datasets by

Value

Returns an `edinfo` class object that can be used to get environmental data with other functions

Author(s)

Taiki Sakai <taiki.sakai@noaa.gov>

Examples

```
## Not run:
# browse the full list (interactive)
edi <- browseEdinfo()

# search for sst datasets (interactive)
edi <- browseEdinfo(var='sst')

## End(Not run)
```

createSSP *Create Sound Speed Profiles*

Description

Creates sound speed profiles (Depth vs Sound Speed) using temperature and salinity data downloaded from HYCOM data servers

Usage

```
createSSP(
  x,
  f = 30000,
  nc = NULL,
  ncVars = c("salinity", "water_temp"),
  dropNA = TRUE,
  progress = TRUE,
  ...
)
```

Arguments

x	a data.frame with columns UTC, Longitude, and Latitude to create sound speed profiles for
f	the frequency (Hz) to generate the profile for
nc	netcdf file containing salinity and temperature data at depth, if NULL (default) these will be downloaded from HYCOM servers
ncVars	names of the salinity and temperature variables (in that order) in your netcdf file, only change these if you are providing your own file to nc
dropNA	logical flag to drop NA values from soundspeeed profile from outputs. SSP will be calculated up to the maximum depth at each coordinate, which can vary. Setting this option to FALSE ensures that outputs are the same length for each coordinate, which can be useful
progress	logical flag to show progress bar for SST download
...	additional arguments to pass to matchEnvData

Value

a list with one element for each row of x, each element is a list containing speed, the sound speed (m/s), and depth (m)

Author(s)

Taiki Sakai <taiki.sakai@noaa.gov>

Examples

```
## Not run:
# examples not run because they require internet connection
coords <- data.frame(UTC=as.POSIXct('2014-07-15 01:00:00', tz='UTC'),
                     Longitude = -119, Latitude = 33)
ssp <- createSSP(coords)
plot(x=ssp[[1]]$speed, y=-ssp[[1]]$depth, type='l')

## End(Not run)
```

`dataToRanges` *Create List of the Ranges of Coordinates*

Description

Creates a named list with the ranges of Longitude, Latitude, and Time (UTC) data for use in functions like formatURL. Can also specify an amount to buffer the min and max values by for each coordinate

Usage

```
dataToRanges(data, buffer = c(0, 0, 0))
```

Arguments

`data`	a data frame with longitude, latitude, and time (UTC) columns
`buffer`	a vector of the amount to buffer the min and max values of Longitude, Latitude, and UTC by (in that order)

Value

a list with the ranges of coordinates for Longitude, Latitude, and UTC. Ranges are listed as c(left, right), so if your data spans across the dateline

Author(s)

Taiki Sakai <taiki.sakai@noaa.gov>

Examples

```
gps <- data.frame(Latitude = c(32, 32.1, 32.2, 32.2, 32.2),
                  Longitude = c(-110, -110.1, -110.2, -110.3, -110.4),
                  UTC = as.POSIXct(c('2000-01-01 00:00:00', '2000-01-01 00:00:10',
                                     '2000-01-01 00:00:20', '2000-01-01 00:00:30',
                                     '2000-01-01 00:00:40')))
dataToRanges(gps)

dataToRanges(gps, buffer = c(.05, .05, 86400))
```

`decimateWavFiles` *Decimate Wave Files*

Description

Decimate a folder of .wav files or a single .wav file to a new sample rate.

Usage

```
decimateWavFiles(inDir, outDir, newSr, progress = TRUE)
```

Arguments

`inDir`	directory of wave files to decimate. Can also be a single .wav file.
`outDir`	directory to write wave files to
`newSr`	sample rate to decimate the files to
`progress`	logical flag to show progress bar

Details

This code is based on R code written by Jay Barlow.

Value

Invisibly returns the names of all files that were successfully decimated

Author(s)

Taiki Sakai <taiki.sakai@noaa.gov>

Examples

```
# one 20kHz wav file is included in package test data
origDir <- system.file('extdata', package='PAMmisc')
decDir <- file.path(tempdir(), 'decSR')
decWavs <- decimateWavFiles(origDir, decDir, 10000)
file.remove(decWavs)
```

`downloadEnv` *Download Environmental Data*

Description

Downloads environmental data matching the coordinates in a set of data

Usage

```
downloadEnv(
  data,
  edinfo,
  fileName = NULL,
  buffer = c(0, 0, 0),
  timeout = 120,
  progress = TRUE,
  ...
)
```

Arguments

`data`	Data containing Longitude, Latitude, and UTC to download matching environmental data for
`edinfo`	either a edinfo object from getEdinfo or erddapToEdinfo or an ERDDAP dataset ID
`fileName`	name of the file to save downloaded data. If left as the default `NULL`, data will be saved to a temporary folder
`buffer`	numeric vector of the amount to buffer the Longitude, Latitude, and UTC coordinates by
`timeout`	number of seconds before timeout stops download attempt
`progress`	logical flag to show download progress
`...`	not used

Value

if download is successful, invisibly returns the filename. If it fails returns `FALSE`.

If successful, the file name of downloaded data. If not, returns `FALSE`

Author(s)

Taiki Sakai <taiki.sakai@noaa.gov>

Examples

```
data <- data.frame(Latitude = 32, Longitude = -117,
                   UTC = as.POSIXct('2000-01-01 00:00:00', tz='UTC'))
## Not run:
# not run because download could take time
# download jplMURSST41 dataset
edi <- erddapToEdinfo('jplMURSST41')
ncFile <- downloadEnv(data, edi, 'sstData.nc')

# browse suggested sst datasets, then download
edi <- browseEdinfo(var='sst')
ncFile <- downloadEnv(data, edi, 'sstData.nc')

## End(Not run)
```

`edinfoToURL` *Create a URL for Downloading Data from a edinfo Object*

Description

Creates a properly formatted URL (see formatURL) from a datalist either from the package's recommended sources or an ERDDAP dataset id

Usage

```
edinfoToURL(edinfo, ranges)
```

Arguments

`edinfo`	a edinfo class object, either from getEdinfo or created by erddapToEdinfo
`ranges`	list of ranges for Longitude, Latitude, and UTC. Must be a named list with a vector of min/max values for each of the three dimensions

Value

a properly formatted URL that can be used to download environmental data

Author(s)

Taiki Sakai <taiki.sakai@noaa.gov>

Examples

```
sstEdi <- getEdinfo()[['jplMURSST41']]
# select all variables for download
sstEdi <- varSelect(sstEdi, TRUE)
edinfoToURL(sstEdi, ranges = list(Latitude = c(32, 33),
                                  Longitude = c(-118, -117),
                                  UTC = as.POSIXct(c('2000-01-01 00:00:00',
                                                     '2000-01-02 00:00:00'), tz='UTC')))
```

`erddapList` *A list of edinfo objects from ERDDAP data sources*

Description

A list of edinfo objects, mostly used internally for functions. These objects represent different environmental data sources from ERDDAP servers and are used to download environmental data.

Usage

```
erddapList
```

Format

A list with objects of class edinfo

Source

Southwest Fisheries Science Center / NMFS / NOAA

`erddapToEdinfo` *Create an edinfo Object from an ERDDAP Dataset Id*

Description

Creates an edinfo object that can be used to create a URL for downloading environmental data using edinfoToURL

Usage

```
erddapToEdinfo(
  dataset,
  baseurl = c("https://upwell.pfeg.noaa.gov/erddap/",
   "https://coastwatch.pfeg.noaa.gov/erddap/", "https://www.ncei.noaa.gov/erddap/",
    "https://erddap.sensors.ioos.us/erddap"),
  chooseVars = TRUE
)
```

```
hycomToEdinfo(
  dataset = "GLBy0.08/expt_93.0",
  baseurl = "https://ncss.hycom.org/thredds/ncss/",
  chooseVars = TRUE
)
```

Arguments

dataset	an ERDDAP or HYCOM dataset id, or the result from info
baseurl	the base URL of an ERDDAP/HYCOM server
chooseVars	logical flag whether or not to select which variables you want now or character vector naming variables to select

Value

an edinfo list object that can be used to download environmental data

Author(s)

Taiki Sakai <taiki.sakai@noaa.gov>

Examples

```
## Not run:
# examples not run because they require internet connection
sstEdi <- erddapToEdinfo('jplMURSST41')
# dataset from a diferent erddap server
sshEdi <- erddapToEdinfo('hawaii_soest_2ee3_0bfa_a8d6',
                         baseurl = 'http://apdrc.soest.hawaii.edu/erddap/')
# THese work the same - erddap function will pass to hycom if appears to be hycom dataset
hycomEdi <- hycomToEdinfo('GLBy0.08/expt_93.0')
hycomEdi <- erddapToEdinfo('GLBy0.08/expt_93.0')

## End(Not run)
```

---

fastReadWave *Load Wave File*

Description

loads data from a WAVE file

Usage

```
fastReadWave(where, from = 0, to = NA_real_, header = FALSE, toWaveMC = FALSE)
```

Arguments

where	file to load data from
from	starting point to load data from (seconds)
to	end point to read data to (seconds), NA to read til end
header	logical flag to read only header information
toWaveMC	logical flag to return a WaveMC object

Details

WAVE is a RIFF (Resource Interchange File Format) widely used for storage of uncompressed audio data. It is often identified by the extension .WAV on DOS-legacy systems (such as Windows). Although WAVE files may contain compressed data, the above functions only support plain, uncompressed PCM data.

This function was originally written Simon Urbanek, all credit for the bulk of the C programming goes to him. Adapted by Taiki Sakai to add additional features and fix some bugs. See additional license comments in file.c

Value

returns an object of the class audioSample as loaded from the WAVE file, or if header=TRUE a named list with the sample.rate, num channels, bit rate, and sample length. audioSample objects store the wav data as a matrix wth one row for every channel and attributes "rate" and "bits"

Author(s)

Taiki Sakai <taiki.sakai@noaa.gov>

---

findEchoTimes *Find Estimated Echo Times*

Description

Finds the estimated times of echoes in a waveform clip. This function was developed to estimate the time of a surface reflected echo of echolocation clicks of deep diving marine mammals. The times of echoes are estimated by finding peaks in the autocorrelation of a signal.

Usage

```
findEchoTimes(
  wav,
  sr = NULL,
  filter = NULL,
  clipLen = 0.03,
  peakMin = 0.01,
  minTime = 0.001,
  maxTime = NULL,
```

```
  channel = NULL,
  n = 3,
  plot = TRUE,
  plotText = NULL
)
```

Arguments

wav	waveform to find echoes in. Can be a numeric vector, Wave, or WaveMC class object
sr	sample rate of the waveform, if wav is a Wave or WaveMC object it will use the samp.rate slot
filter	filter to apply to wav, a vector of two numbers specifying the lower and upper bounds of the filter in Hz. A first value of 0 means no highpass filter is applied, a second value greater than sr/2 means no lowpass filter is applied.
clipLen	length of clip (seconds) to analyse for echoes, measured from start of wav
peakMin	minimum magnitude of autocorrelation value to be considered a possible peak
minTime	minimum allowed echo time (seconds), this should be large enough to avoid correlating the original pulse with itself
maxTime	maximum allowed echo time (seconds)
channel	if wav has multiple channels, channel to use
n	the number of potential echoes to return, times with the n highest autocorrelation magnitude will be returned
plot	logical flag to create plot, will create a two-panel plot of the waveform (top) and the autocorrelation (bottom). Points of the selected candidate echo times are also drawn
plotText	optional text to plot on the upper waveform plot

Value

a list with elements mag, time and wav

Author(s)

Taiki Sakai <taiki.sakai@noaa.gov>

---

formatURL *Format URL for Environmental Data Download*

Description

This creates a properly formatted URL for downloading environmental data either from an ERD-DAP or HYCOM server. This URL can be pasted into a browser or submitted to something like httr::GET to actually download the data. Also see edinfoToURL

Usage

```
formatURL(
  base,
  dataset,
  fileType,
  vars,
  ranges,
  stride = 1,
  style = c("erddap", "hycom")
)
```

Arguments

base	the base URL to download from
dataset	the specific datased ID to download
fileType	the type of file to download, usually a netcdf
vars	a vector of variables to download
ranges	a list of three vectors specifying the range of data to download, must a list with named vectors Longitude, Latitude, and UTC where each vector is c(min, max) (Note: even if the time is something like "dayOfYear" this should still be called 'UTC' for the purpose of this list). (see dataToRanges).
stride	the stride for all dimensions, a value of 1 gets every data point, 2 gets every other, etc.
style	either 'erddap' or 'hycom'

Value

a properly formatted URL that can be used to download environmental data

Author(s)

Taiki Sakai <taiki.sakai@noaa.gov>

Examples

```
formatURL(
    base = "https://upwell.pfeg.noaa.gov/erddap/griddap/",
    dataset = "jplMURSST41",
    fileType = "nc",
    vars = "analysed_sst",
    ranges = list(
               Latitude = c(30, 31),
               Longitude = c(-118, -117),
              UTC = as.POSIXct(c('2005-01-01 00:00:00', '2005-01-02 00:00:00'), tz='UTC')
             ),
    stride=1,
    style = 'erddap'
)
```

getEdinfo *Browse a List of Curated Environmental Datasets*

Description

This function gets the list of environmental datasets provided as a recommended starting point for various measures

Usage

```
getEdinfo()
```

Value

a list of edinfo list objects

Author(s)

Taiki Sakai <taiki.sakai@noaa.gov>

Examples

```
ediList <- getEdinfo()
ediList[[1]]
ediList[['jplMURSST41']]
```

getFigshareInfo *getFigshareInfo*

Description

downloads filename and recording URL information from a Figshare article. Requires a users API token from their figshare account

Usage

```
getFigshareInfo(token, id)
```

Arguments

token	Personal API token from users Figshare account, see here for information on creating a token
id	Figshare article ID to download information for

Value

dataframe with columns `filename` and `recording_url`

Author(s)

Taiki Sakai <taiki.sakai@noaa.gov>

`hycomList` *A list of edinfo objects from HYCOM data sources*

Description

A list of edinfo objects, mostly used internally for functions. These objects represent different environmental data sources from HYCOM servers and are used to download environmental data.

Usage

```
hycomList
```

Format

A list with objects of class edinfo

Source

Southwest Fisheries Science Center / NMFS / NOAA

`matchEnvData` *Match Data From an Existing Netcdf File or Download and Match*

Description

Extracts all variables from a netcdf file matching Longitude, Latitude, and UTC coordinates in given dataframe

Usage

```
matchEnvData(
  data,
  nc = NULL,
  var = NULL,
  buffer = c(0, 0, 0),
  FUN = c(mean),
  fileName = NULL,
  progress = TRUE,
```

```
  depth = 0,
  ...
)

## S4 method for signature 'data.frame'
matchEnvData(
  data,
  nc = NULL,
  var = NULL,
  buffer = c(0, 0, 0),
  FUN = c(mean),
  fileName = NULL,
  progress = TRUE,
  depth = 0,
  ...
)
```

Arguments

`data`	dataframe containing Longitude, Latitude, and UTC to extract matching variables from the netcdf file
`nc`	name of a netcdf file, ERDDAP dataset id, or an edinfo object
`var`	(optional) vector of variable names
`buffer`	vector of Longitude, Latitude, and Time (seconds) to buffer around each datapoint. All values within the buffer will be used to report the mean, median, and standard deviation
`FUN`	a vector or list of functions to apply to the data. Default is to apply mean, median, and standard deviation calculations
`fileName`	(optional) file name to save downloaded nc file to. If not provided, then no nc files will be stored, instead small temporary files will be downloaded and then deleted. This can be much faster, but means that the data will need to be downloaded again in the future. If `fileName` is provided, then the function will attempt to download a single nc file covering the entire range of your data. If your data spans a large amount of time and space this can be problematic.
`progress`	logical flag to show progress bar
`depth`	depth values (meters) to use for matching, overrides any `Depth` column in the data or can be used to specify desired depth range when not present in data. Variables will be summarised over the range of these depth values. `NULL` uses all available depth values
`...`	other parameters to pass to ncToData

Value

original dataframe with three attached columns for each variable in the netcdf file, one for each of mean, median, and standard deviation of all values within the buffer

Author(s)

Taiki Sakai <taiki.sakai@noaa.gov>

Examples

```
data <- data.frame(Latitude = 32, Longitude = -117,
                   UTC = as.POSIXct('2004-12-31 09:00:00', tz='UTC'))
## Not run:
# Not run because downloads files
# default calculates mean, median, and standard deviation
matchEnvData(data, nc='jplMURSST41', var=c('analysed_sst', 'analysis_error'))
# get just mean within a buffer around coordinates
matchEnvData(data, nc='jplMURSST41', var=c('analysed_sst', 'analysis_error'),
             FUN = mean, buffer = c(.01, .01, 86400))

## End(Not run)
# Can also work from an existing nc file
nc <- system.file('extdata', 'sst.nc', package='PAMmisc')
matchEnvData(data, nc = nc)
# Using a custom function
meanPlusOne <- function(x) {
  mean(x, na.rm=TRUE) + 1
}
matchEnvData(data, nc=nc, FUN=c(mean, meanPlusOne))
```

---

ncToData *Match Data From a Netcdf File*

Description

Extracts all variables from a netcdf file matching Longitude, Latitude, and UTC coordinates in given dataframe

Usage

```
ncToData(
  data,
  nc,
  var = NULL,
  buffer = c(0, 0, 0),
  FUN = c(mean),
  raw = FALSE,
  keepMatch = TRUE,
  progress = TRUE,
  depth = 0,
  verbose = TRUE,
  ...
)
```

Arguments

data	dataframe containing Longitude, Latitude, and UTC to extract matching variables from the netcdf file
nc	name of a netcdf file
var	(optional) character vector of variable names to match. If NULL, all variables present in nc will be used
buffer	vector of Longitude, Latitude, and Time (seconds) to buffer around each datapoint. All values within the buffer will be used to report the mean, median, and standard deviation
FUN	a vector or list of functions to apply to the data. Default is to apply mean, median, and standard deviation calculations
raw	logical flag to return only the raw values of the variables. If TRUE the output will be changed to a list with length equal to the number of data points. Each item in the list will have separate named entries for each variable that will have all values within the given buffer and all values for any Z coordinates present.
keepMatch	logical flag to keep the matched coordinates, these are useful to make sure the closest point is actually close to your XYZT
progress	logical flag to show progress bar for matching data
depth	depth values (meters) to use for matching, overrides any Depth column in the data or can be used to specify desired depth range when not present in data. Variables will be summarised over the range of these depth values. NULL uses all available depth values
verbose	logical flag to show warning messages for possible coordinate mismatch
...	not used

Value

original dataframe with three attached columns for each variable in the netcdf file, one for each of mean, median, and standard deviation of all values within the buffer

Author(s)

Taiki Sakai <taiki.sakai@noaa.gov>

Examples

```
data <- data.frame(Latitude = 32, Longitude = -117,
                   UTC = as.POSIXct('2005-01-01 00:00:00', tz='UTC'))
nc <- system.file('extdata', 'sst.nc', package='PAMmisc')
# default calculates mean
ncToData(data, nc = nc)
# calculate mean, median, and sd
ncToData(data, nc=nc, FUN=c(mean, median, sd), buffer = c(.01, .01, 86400))
# custom function
meanPlusOne <- function(x) {
    mean(x, na.rm=TRUE) + 1
```

```
}
ncToData(data, nc=nc, FUN=c(mean, meanPlusOne))
```

---

peakTrough *Find Peaks and Troughs in a Spectrum*

Description

Finds up to three peaks in a spectrum, as well as the troughs between those peaks.

Usage

```
peakTrough(spec, freqBounds = c(10, 30), dbMin = -15, smooth = 5, plot = FALSE)
```

Arguments

spec	the spectrum of a signal, the first column must be frequency in kilohertz, the second column must be dB
freqBounds	a two element vector specifying the frequency range around the highest peak to search for a second/third peak. Units are in kHz, a value of c(f1, f2) requires a second peak to be at least f1 kHz away from the first peak, but no further than f2 kHz away.
dbMin	minimum dB level for second / third peaks, relative to maximum dB. Any points lower than this dB level will not be considered a candidate peak.
smooth	the amount to smooth the spectrum before attempting to find second / third peaks. Uses a simple local average, smooth is the total number of points to use. A value of 1 applies no smoothing.
plot	logical flag to plot image of peak/trough locations on spectrum. Useful for finding appropriate settings for freqBounds and dbMin

Details

The first peak is the frequency with the highest dB level (first and last frequency points are ignored). Then this uses a very simple algorithm to find second and third peaks in a spectrum. Peak candidates are identified with a few simple steps:

Step 1 Use a local average of (smooth) points to smooth the spectrum.

Step 2 Check if a point is larger than both its neighbors.

Step 3 Check if points are within the frequency range specified by freqBounds. Points must be at least f1 kHz away from the frequency , but no further than f2 kHz away.

Step 4 Check if points are above the minimum dB level specified by dbMin.

From the remaining points the point with the highest dB level is selected as the second peak, then the frequency range filter of Step 3 is applied again around this second peak before attempting to find a third peak. If no second or third peak is found (ie. no values fall within the specified frequency and dB search ranges), then it will be set to 0. The trough values are set as the frequency with the lowest dB level between any peaks that were found. The trough values will be 0 for any peaks that were not found.

If you are unsure of what levels to specify for `freqBounds` and `dbMin`, setting `plot=TRUE` will show a visualization of the search range and selected peaks so you can easily see if the selected parameters are capturing the behavior you want.

Value

a dataframe with the frequencies (in kHz) of up to 3 peaks and 2 troughs between those peaks. Also reports the peak-to-peak distance. Any peaks / troughs that were not able to be found (based on `freqBounds` and `dbMin` parameters) will be 0.

Author(s)

Taiki Sakai <taiki.sakai@noaa.gov>

Examples

```
clickWave <- createClickWave(signalLength = .1, clickLength = 1000, clicksPerSecond = 200,
                             frequency = 3e3, sampleRate = 10e3)
peakTrough(seewave::spec(clickWave, plot=FALSE), plot=TRUE)
```

`plotPresBar` *plotPresBar*

Description

Creates a bar plot of the presence or density of detections across time

Usage

```
plotPresBar(
  x,
  start = NULL,
  end = NULL,
  bin = "hour/day",
  by = NULL,
  title = TRUE,
  fill = "grey35",
  format = c("%m/%d/%Y %H:%M:%S", "%m-%d-%Y %H:%M:%S",
    "%Y/%m/%d %H:%M:%S", "%Y-%m-%d %H:%M:%S"),
  plotTz = "UTC"
)
```

Arguments

x	a data.frame of detections, must have a column UTC that contains the time of detection as a POSIXct object in UTC timezone
start	the beginning datetime of the plot, if NULL will be set to the minimum time in x
end	the ending datetime of the plot, if NULL will be set to the maximum time in x
bin	string identifying how to bin detections. Acceptable time units are c('minute', 'hour', 'day', 'week', 'month'). For presence, bin should be of the form 'unit1/unit2', e.g. 'hour/day' will show the hours per day with detections. For call density, bin is a single time unit, e.g. 'hour' will show the number of calls per hour. Call density can also be specified as 'call/hour'. Note that plural forms of all units are accepted.
by	(optional) if not NULL, specifies the name of a column in x to split and color the bars by
title	if TRUE, a title will automatically created. If any other value, that will be used for the title of the plot.
fill	the fill color for the bars, only used if by is NULL, otherwise bars are colored by species using the default ggplot2 palette
format	date format if UTC column of x is a character
plotTz	the timezone to use for plotting the data. Note that inputs must still be in UTC, this option allows you to create plots scaled to local time. Valid values come from OlsonNames

Value

a ggplot2 object

Author(s)

Taiki Sakai <taiki.sakai@noaa.gov>

Examples

```
df <- data.frame(UTC = as.POSIXct(runif(1e2, min=0, max=7*24*3600),
                                  origin='1970-01-01 00:00:00', tz='UTC'),
                 label = sample(letters[1:3], 1e2, replace=TRUE))
# hours per day with detections
plotPresBar(df, bin='hour/day')
# calls per day - these options are identical
plotPresBar(df, bin='day')
plotPresBar(df, bin='call/day')
plotPresBar(df, bin='calls/day')
# calls per day, colored by 'label'
plotPresBar(df, bin='day', by='label')
```

plotPresGrid *plotPresGrid*

Description

Creates a grid plot of the presence or density of detections across time where the x-axis is the hour of the day and the y-axis is the date

Usage

```
plotPresGrid(
  x,
  start = NULL,
  end = NULL,
  bin = c("hour", "minute", "30min", "15min"),
  type = c("presence", "density"),
  by = NULL,
  alpha = 0.5,
  gps = NULL,
  format = c("%m/%d/%Y %H:%M:%S", "%m-%d-%Y %H:%M:%S",
    "%Y/%m/%d %H:%M:%S", "%Y-%m-%d %H:%M:%S"),
  fill = "blue",
  color = NA,
  cmap = viridis_pal()(25),
  title = TRUE,
  plotTz = "UTC"
)
```

Arguments

x	a data.frame of detections, must have a column UTC that contains the time of detection as a POSIXct object in UTC timezone
start	the beginning datetime of the plot, if NULL will be set to the minimum time in x
end	the ending datetime of the plot, if NULL will be set to the maximum time in x
bin	the unit of time for each rectangle in the grid, must be one of "hour", "minute", "30min", or "15min"
type	one of either "presence" or "density". If "density", then boxes will be colored according to the number of detections in each timeBin will be plotted. If "presence", then each box will be colored by fill
by	(optional) if not NULL, specifies the name of a column in x to split and color the rectangles by. Only valid for presence plots.
alpha	opacity of rectangles, only used if by is not NULL
gps	(optional) if not NULL, a data.frame of GPS coordinates covering the date range of x. These are used to calculate sunrise and sunset information which is shown as a shaded dark region in the background of the plot. The data.frame must

	have columns "UTC", "Latitude", and "Longitude". If columns "Latitude" and "Longitude" are present in x, then these values will be used and you do not need to provide separate GPS data here
format	date format if UTC column of x is a character
fill	the fill color for the boxes, only used if type is "presence"
color	the outline color for the boxes, only used if type is "presence"
cmap	the colormap to use for the boxes, only used if type is "density"
title	if TRUE, a title will automatically created. If any other value, that will be used for the title of the plot.
plotTz	the timezone to use for plotting the data. Note that inputs must still be in UTC, this option allows you to create plots scaled to local time. Valid values come from OlsonNames

Value

a ggplot2 object

Author(s)

Taiki Sakai <taiki.sakai@noaa.gov>

Examples

```
df <- data.frame(UTC = as.POSIXct(runif(1e2, min=0, max=7*24*3600),
                                  origin='1970-01-01 00:00:00', tz='UTC'),
                 label = sample(letters[1:3], 1e2, replace=TRUE))
plotPresGrid(df, type='presence', bin='hour')
plotPresGrid(df, type='density', bin='hour')
plotPresGrid(df, type='density', bin='30min')
gps <- data.frame(UTC = as.POSIXct('1970-01-01 00:00:00', tz='UTC'),
                 Latitude=32.4,
                 Longitude = -118)
plotPresGrid(df, gps=gps, bin='hour')
# coloring presence grid by label column
plotPresGrid(df, gps=gps, by='label')
# can be confusing if there is a lot of overlap, ggplot output can be split
library(ggplot2)
plotPresGrid(df, gps=gps, by='label') + facet_wrap(vars(label), ncol=2)
# using "by" with type="density" defaults to this facet_wrap behavior
# since color is already being used to represent density
plotPresGrid(df, gps=gps, by='label', type='density')
# can adjust facet_wrap parameters by adding it again
plotPresGrid(df, gps=gps, by='label', type='density') + facet_wrap(vars(label), ncol=2)
```

pwelch *Estimate Power Spectral Density Using Welch's Method*

Description

Estimates the power spectral density (PSD) of an input signal using Welch's method. This should function similarly to the Matlab function pwelch, but results may not be identical. Breaks the input signal into (usually) overlapping frames and averages the resulting PSD estimates

Usage

```
pwelch(
  x,
  nfft,
  noverlap = 0,
  sr = NULL,
  window = NULL,
  demean = c("long", "short", "none"),
  channel = 1
)
```

Arguments

x	input signal, either a numeric vector, Wave, WaveMC, or audioSample object. Can also be a path to a wav file and it will be read in
nfft	length of FFT window to use for individual frames
noverlap	number of samples each frame should overlap
sr	sample rate of data, only necessary if x is a vector
window	window to apply, must be a vector of length nfft. If NULL (default), then a hamming window will be used
demean	method of demeaning the signal, one of 'long', 'short', or 'none'. Long subtracts the mean of the entire signal x, short subtracts the mean of each individual frame, none does no mean subtraction.
channel	channel number to analyse, ignored if x is a vector

Value

returns a list with items spec, the PSD estimate of the input signal, and freq, the frequency values (Hz) at each value of spec

Author(s)

Taiki Sakai <taiki.sakai@noaa.gov>

Examples

```
# wav example is synthetic echolocation clicks at 4kHz
wavFile <- system.file('extdata/testWav.wav', package='PAMmisc')
psd <- pwelch(wavFile, nfft=1e3, noverlap=500, demean='long')
plot(x=psd$freq, y=10*log10(psd$spec), type='l')
```

---

raytrace *Raytrace Through a Soundspeed Profile*

Description

Traces the ray of a sound through a varying soundspeed profile for a fixed amount of time. Also plots the provided sound speed profile and all traces generated. All code here is based on MATLAB code originally written by Val Schmidt from the University of New Hampshire Val Schmidt (2021). raytrace https://www.mathworks.com/matlabcentral/fileexchange/26253-raytrace), MATLAB Central File Exchange. Retrieved June 29, 2021.

Usage

```
raytrace(x0, z0, theta0, tt, zz, cc, plot = TRUE, progress = FALSE)
```

Arguments

x0	starting horizontal coordinate in meters
z0	starting vertical coordinate in meters
theta0	starting angle(s) of ray in degrees
tt	max travel time of ray in seconds
zz	vertical coordinates of sound speed profile (positive values are down)
cc	sound speed measurements at zz locations, meters / second
plot	logical flag to plot. Can be a vector of length two to individually select plotting one of the two plots generated
progress	logical flag to show progress bar

Value

A list with four elements: x, the horizontal coordinates of ray path, z the vertical coordinates of ray path, t actual travel time of ray in seconds, and d the total distance the ray traveled. Each individual item in the output is a list with one entry for each theta0 provided.

Author(s)

Taiki Sakai <taiki.sakai@noaa.gov>

Examples

```
#  Setup the sound speed profile
zz <- seq(from=0, to=5000, by=1)
cc <- 1520 + zz * -.05
cc[751:length(cc)] <- cc[750] + (zz[751:length(zz)] - zz[750])*.014
rt <- raytrace(0, 0, 5, 120, zz, cc, TRUE)
```

---

readGPXTrack *Read Tracks from a GPX File*

Description

Read in a GPX file and convert the tracks to a dataframe

Usage

```
readGPXTrack(x)
```

Arguments

x	a path to a .gpx file

Value

a dataframe with columns Latitude, Longitude, UTC, and Name

Author(s)

Taiki Sakai <taiki.sakai@noaa.gov>

Examples

```
gpxFile <- system.file('extdata', 'GPX.gpx', package='PAMmisc')
gpxData <- readGPXTrack(gpxFile)
str(gpxData)
```

readSpecAnno *Read Pamguard Spectrogram Annotation Table*

Description

Reads the Spectrogram Annotation table from a PAMGuard database and applies some minor formatting

Usage

```
readSpecAnno(db, table = "Spectrogram_Annotation")
```

Arguments

db	database file to read data from
table	name of the Spectrogram Annotation table to read

Value

a dataframe containing spectrogram annotation data

Author(s)

Taiki Sakai <taiki.sakai@noaa.gov>

Examples

```
## Not run:
myDb <- 'PamguardDatabase.sqlite3'
specAnno <- readSpecAnno(db)

## End(Not run)
```

soundtrapQAQC *Perform QA/QC on Soundtrap Files*

Description

Gathers data from Soundtrap XML log files to perform QA/QC on a set of recordings.

Usage

```
soundtrapQAQC(
  dir,
  outDir = NULL,
  xlim = NULL,
  label = NULL,
  voltSelect = c("internal", "external"),
  plot = TRUE
)

processSoundtrapLogs(dir, voltSelect = c("internal", "external"))
```

Arguments

dir	directory containing Soundtrap XML logs, wav files, and SUD files. Can either be a single directory containing folders with all files (will search recursively), or a vector of three directories containing the SUD files, wav files, and XML files (in that order - alphabetical S-W-X)
outDir	if provided, output plots and data will be written to this folder
xlim	date limit for plots
label	label to be used for plots and names of exported files
voltSelect	one of "internal" or "external" to select which battery voltage to use
plot	logical flag to create output plots

Value

list of dataframes with summary data for `$xmlInfo`, `$sudInfo`, and `$wavInfo`

Author(s)

Taiki Sakai <taiki.sakai@noaa.gov>

Examples

```
## Not run:
# not run
stDir <- './Data/SoundtrapFiles/'
stData <- soundtrapQAQC(stDir, plot=TRUE)
# save data
stData <- soundtrapQAQC(stDir, outDir='./Data/SoundtrapFiles/QAQC', plot=TRUE)
# or provide separate folders of data
stDirs <- c('./Data/SoundtrapFiles/SUDFiles',
            './Data/SoundtrapFiles/WavFiles',
            './Data/SoundtrapFiles/XMLFiles')
stData <- soundtrapQAQC(stDirs, plot=TRUE)

## End(Not run)
```

squishList *Compress a List by Name*

Description

Attempts to compress a list by combining elements with the same name, searching recursively if there are lists in your list

Usage

```
squishList(myList, unique = FALSE)
```

Arguments

myList	a list with named elements to be compressed
unique	logical flag to try and reduce result to only unique values

Details

items with the same name are assumed to have the same structure and will be combined. Dataframes will be combined with bind_rows, vectors just be collapsed into one vector, matrices will be combined with rbind, lists will be combined recursively with another call to `squishList`

Value

a list with one element for every unique name in the original list

Author(s)

Taiki Sakai <taiki.sakai@noaa.gov>

Examples

```
myList <- list(a=1:3, b=letters[1:4], a=5:6, b=letters[4:10])
squishList(myList)

myList <- list(a=1:3, b=data.frame(x=1:3, y=4:6), b=data.frame(x=10:14, y=1:5))
squishList(myList)

myList <- list(a=list(c=1:2, d=2), b=letters[1:3], a=list(c=4:5, d=6:9))
squishList(myList)
```

straightPath *Mark Straight Path Segments in GPS Track*

Description

This function attempts to mark portions of a GPS track where a ship is traveling in a straight line by comparing the recent average heading with a longer term average heading. If these are different, then the ship should be turning. Note this currently does not take in to account time, only number of points

Usage

```
straightPath(gps, nSmall = 10, nLarge = 60, thresh = 10, plot = FALSE)
```

Arguments

gps	gps data with columns Longitude, Latitude, and UTC (POSIX format). Usually this has been read in from a Pamguard database, in which case columns Heading and Speed will also be used.
nSmall	number of points to average to get ship's current heading
nLarge	number of points to average to get ship's longer trend heading
thresh	the amount which nSmall and nBig should differ by to call this a turn
plot	logical flag to plot result, gps must also have columns Latitude and Longitude

Value

the original dataframe gps with an added logical column straight indicating which portions are approximately straight

Author(s)

Taiki Sakai <taiki.sakai@noaa.gov>

Examples

```
gps <- data.frame(Latitude = c(32, 32.1, 32.2, 32.2, 32.2),
                  Longitude = c(-110, -110.1, -110.2, -110.3, -110.4),
                  UTC = as.POSIXct(c('2000-01-01 00:00:00', '2000-01-01 00:00:10',
                                     '2000-01-01 00:00:20', '2000-01-01 00:00:30',
                                     '2000-01-01 00:00:40')),
                  Heading = c(320, 320, 270, 270, 270),
                  Speed = c(.8, .8, .5, .5, .5))

straightPath(gps, nSmall=1, nLarge=2)

straightPath(gps, nSmall=1, nLarge=4)
```

`trainSplitPermute` *trainSplitPermute*

Description

Find a desired train/val/test split of a dataset through random permutation. Uses a variable in your dataset to randomly split by (for example, could be the location of different sites, or different months of data), then tries to find the split that most closesly matches your desired distribution of data for a set of labels. It can often be difficult to find a good split if the distribution of your labels is not consistent across sites, so this function tries a bunch of random splits then uses a score to find the best one.

Usage

```
trainSplitPermute(
  x,
  probs = c(0.7, 0.15, 0.15),
  n = 1000,
  splitBy = "drift",
  label = "species",
  countCol = NULL,
  minCount = c(1, 1, 1),
  top = 3,
  seed = 112188
)
```

Arguments

`x`	a dataframe of data you want to find splits for
`probs`	a vector of 3 values that sum to one defining what percentage of data should be in your training, validation, and test sets (respectively)
`n`	number of random samples to try. If your labels are fairly evenly distributed this can be smaller, but needs to be larger for more uneven distributions
`splitBy`	name of column containing the variable you want to split by
`label`	name of the column containing your dataset labels
`countCol`	the names of any additional columns in your dataset defining the quantities you want to count (see example for why this is useful)
`minCount`	minimum count for each split category, usually safe to leave this as the default of 1 for all splits
`top`	the number of results to return. Usually you want to use just the best scoring result, but this can occasionally result in splits that are distributed in an undesirable way by random chance (eg maybe all sites in your validation data are unintentionally clustered together)
`seed`	random seed to set for reproducibility

Value

a list of the top results. Each individual result contains $splitMap containing the random split marked as integer 1, 2, 3 corresponding to train, val, test and $splitVec a vector marking each row of x with its category. These two results are named by the levels of splitBy. $distribution a table of the distribution of label in the split, and $score the split score (lower is closer to desired probs)

Author(s)

Taiki Sakai <taiki.sakai@noaa.gov>

Examples

```
# making some dummy data
df <- data.frame(
    species = sample(letters[1:5], prob=c(.4, .2, .1, .1, .2), 1e3, replace=TRUE),
    site = sample(LETTERS[1:12], 1e3, replace=TRUE),
    event = 1:1e3
)
# try a split with n=3
split <- trainSplitPermute(df, probs=c(.7, .15, .15), n=3, label='species', splitBy='site')
# assign the best split as the split cateogry
df$split <- split[[1]]$splitVec
# distribution is not close to our desired .7, .15, .15 split because n is too low
round(table(df$species, df$split) /
    matrix(rep(table(df$species), 3), nrow=5), 2)

# rerun with higher n to get closer to desired distribution
split <- trainSplitPermute(df, probs=c(.7, .15, .15), n=1e3, label='species', splitBy='site')
df$split <- split[[1]]$splitVec
round(table(df$species, df$split) /
    matrix(rep(table(df$species), 3), nrow=5), 2)

# adding a new site that has significantly more detections than others
addSite <- data.frame(
    species = sample(letters[1:5], 500, replace=TRUE),
    site = rep(LETTERS[13], 500),
    event = 1001:1500)
df$split <- NULL
df <- rbind(df, addSite)

# now just splitting by site does not result in a balanced split for our number of species
# it splits the sites to approx .7, .15, .15 but this does not result in balanced species
split <- trainSplitPermute(df, probs=c(.7, .15, .15), n=1e3, label='species', splitBy='site')
df$split <- split[[1]]$splitVec
round(table(df$species, df$split) /
    matrix(rep(table(df$species), 3), nrow=5), 2)

# adding 'event' as a countCol fixes this
split <- trainSplitPermute(df, probs=c(.7, .15, .15), n=1e3, label='species',
    splitBy='site', countCol='event')
df$split <- split[[1]]$splitVec
```

```
round(table(df$species, df$split) /
    matrix(rep(table(df$species), 3), nrow=5), 2)
```

---

updateUID *Update Detection UIDs*

Description

Update the UIDs of detections in a Pamguard database. UIDs can become mismatched when re-running data, this will attempt to re-associate the new UIDs in binary files with detections in the database

Usage

```
updateUID(db, binaries, verbose = TRUE, progress = TRUE)
```

Arguments

db	database file to update UIDs
binaries	folder of binary files to use for updating
verbose	logical flag to show summary messages
progress	logical flag to show progress bars

Value

Same database as db, but with an additional column "newUID" added to each detection table with updated UIDs if found. "newUID" will be -1 for any detections where no match was found

Author(s)

Taiki Sakai <taiki.sakai@noaa.gov>

Examples

```
## Not run:
# not run because sample data does not exist
db <- 'MismatchedUid.sqlite3'
bin <- './BinaryFolder'
updateUID(db, bin)

## End(Not run)
```

varSelect *Utility for Selecting Variables to Download*

Description

Loops through the available variables in an edinfo object and asks whether or not each should be downloaded, then stores the result for passing on to formatURL

Usage

```
varSelect(edinfo, select = NULL)
```

Arguments

edinfo	a datalist, either from getEdinfo or created by erddapToEdinfo
select	(optional) logical vector of which variables to select. If left as default NULL, user will be prompted to select which variables to keep. If not NULL, can either be a single TRUE to select all variables, or a logical vector of length equal to the number of variables in edinfo. Can also be a vector of variable names to select.

Value

the same object as edinfo with an updated varSelect field

Author(s)

Taiki Sakai <taiki.sakai@noaa.gov>

Examples

```
sstEdi <- getEdinfo()[['jplMURSST41']]
## Not run:
# interactively select
sstEdi <- varSelect(sstEdi)

## End(Not run)
# select all variables
sstEdi <- varSelect(sstEdi, TRUE)
# select the first two of four
sstEdi <- varSelect(sstEdi, c(TRUE, TRUE, FALSE, FALSE))
```

`wignerTransform` *Calculate the Wigner-Ville Transform of a Signal*

Description

Calculates the Wigner-Ville transform a signal. By default, the signal will be zero-padded to the next power of two before computing the transform, and creates an NxN matrix where N is the zero-padded length. Note that this matrix can get very large for larger N, consider shortening longer signals.

Usage

```
wignerTransform(signal, n = NULL, sr, plot = FALSE)
```

Arguments

`signal`	input signal waveform
`n`	number of frequency bins of the output, if NULL will be the next power of two from the length of the input signal (recommended)
`sr`	the sample rate of the data
`plot`	logical flag whether or not to plot the result

Details

This code mostly follows Pamguard's Java code for computing the Wigner-Ville and Hilbert transforms.

Value

a list with three items. `tfr`, the real values of the wigner transform as a matrix with `n` rows and number of columns equal to the next power of two from the length of the input signal. `f` and `t` the values of the frequency and time axes.

Author(s)

Taiki Sakai <taiki.sakai@noaa.gov>

Examples

```
clickWave <- createClickWave(signalLength = .05, clickLength = 1000, clicksPerSecond = 200,
                             frequency = 3e3, sampleRate = 10e3)
wt <- wignerTransform(clickWave@left, n = 1000, sr = 10e3, plot=TRUE)
```

writeAMWave *Write Amplitude Modulated Waveform*

Description

Write a wave file for a synthesized amplitude modulated call

Usage

```
writeAMWave(
  fileName,
  outDir,
  signalLength,
  modFrequency,
  frequency,
  sampleRate,
  window = c(0.55, 0.45),
  silence = c(0, 0),
  gainFactor = 0.1
)

createAMWave(
  signalLength,
  modFrequency,
  frequency,
  sampleRate,
  window = c(0.55, 0.45),
  silence = c(0, 0),
  gainFactor = 0.1
)
```

Arguments

fileName	name of the file to write. If missing, the file be named usign signalLength, modFrequency, frequency, and sampleRate
outDir	directory to write wave files to
signalLength	length of signal to create in seconds
modFrequency	modulation frequency in Hz of the amplitude modulation
frequency	frequency of the AM call
sampleRate	sample rate for the wave file to create
window	window constants for applying the amplitude modulation. See details.
silence	silence to pad before and after signal in seconds
gainFactor	scaling factor between 0 and 1. Low numbers are recommended (default 0.1)

Details

Amplitude modulated signals are modelled as an ideal sinusoid multiplied by a window function. The window function is an offset sinusoid with frequency equal to the modulation frequency:

$$W = .5 + .45 * sin(2\pi mft)$$

See `example(writeAMWave)` for a plot showing how this works.

Value

`writeAMWave` invisibly returns the file name, `createAMWave` returns a Wave class object

Author(s)

Taiki Sakai <taiki.sakai@noaa.gov>

Examples

```
# Visualisation of modelled AM wave
signal <- sin(2*pi*100*(1:1000)/1000)
window <- .55 + .45 * sin(2*pi*15*(1:1000)/1000)
oldMf <- par()$mfrow
par(mfrow=c(3,1))
plot(signal, type='l')
plot(window, type='l')
plot(window*signal, type='l')
tmpFile <- file.path(tempdir(), 'tempWav.wav')
writeAMWave(tmpFile, signalLength = 1, modFrequency = 1000,
               frequency = 30000, sampleRate = 100000)
file.remove(tmpFile)
amWave <- createAMWave(signalLength = 1, modFrequency = 1000,
                            frequency = 30e3, sampleRate = 100e3)
par(mfrow=oldMf)
```

`writeClickWave` *Write Click Waveform*

Description

Write a wave file for a synthesized delphinid click

Usage

```
writeClickWave(
  fileName,
  outDir,
  signalLength,
  clickLength,
  clicksPerSecond,
```

```
  frequency,
  sampleRate,
  silence = c(0, 0),
  gainFactor = 0.1
)

createClickWave(
  signalLength,
  clickLength,
  clicksPerSecond,
  frequency,
  sampleRate,
  silence = c(0, 0),
  gainFactor = 0.1
)
```

Arguments

fileName	name of the file to write. If missing, the file be named usign signalLength, clickLength, clicksPerSecond, frequency, and sampleRate
outDir	directory to write wave files to
signalLength	length of signal to create in seconds
clickLength	length of each click in microseconds
clicksPerSecond	number of clicks per second
frequency	frequency of the clicks
sampleRate	sample rate for the wave file to create
silence	silence to pad before and after signal in seconds
gainFactor	scaling factor between 0 and 1. Low numbers are recommended (default 0.1)

Details

This code is based on Matlab code by Julie Oswald (2004). Clicks are simulated as an exponentially damped sinusoid.

Value

writeClickWave invisibly returns the file name, createClickWave returns a Wave class object

Author(s)

Taiki Sakai <taiki.sakai@noaa.gov>

Examples

```
tmpFile <- file.path(tempdir(), 'tempWav.wav')
writeClickWave(tmpFile, signalLength = 1, clickLength = 100, clicksPerSecond = 200,
               frequency = 30000, sampleRate = 100000)
file.remove(tmpFile)
clickWave <- createClickWave(signalLength = 1, clickLength = 100, clicksPerSecond = 200,
                             frequency = 30e3, sampleRate = 100e3)
```

Index