Package 'glatos'

February 26, 2017

Type Package

Title glatos: A package for the Great Lakes Acoustic Telemetry Observation System

Version 0.1

Date 2017-02-16

Authors Chris Holbrook, Todd Hayden, Tom Binder

Maintainer Chris Holbrook <cholbrook@usgs.gov>

Description Functions useful to members of the Great Lakes Acoustic Telemetry Observation System www.glatos.glos.us; many more broadly relevant to simulating, processing, analysing, and visualizing acoustic telemetry data.

Depends R (>= 3.1.1)

License GPL (2.0 or later)

LazyLoad yes

RoxygenNote 6.0.0

R topics documented:

abacusPlot *Plot detection locations of acoustic transmitters over time*

Description

Plot detection locations of acoustic transmitters over time

Usage

```
abacusPlot(detections, detColNames = list(locationCol = "glatos_array",
  timestampCol = "detection_timestamp_utc"), controlTable = NULL,
  plotTitle = "", Ylab = NA, outFile = "AbacusPlot.png", ...)
```

Arguments

detections	A data frame containing at least two columns with names specified by detColNames. The 'location' column contains the locations (typically 'glatos_array' or 'station' for GLATOS data) that will be plotted on the y-axis. The 'timestamp' column contains the datetime stamps for the detections (MUST be of class 'POSIXct').
detColNames	An optional list of character strings with names of required columns in detections: • locationCol A character scalar with the name (in quotes) of the column containing the location codes to be plotted on the y-axis. The default value ("glatos_array") is consistent with GLATOS standard. • timestampCol A character scalar with the name (in quotes) of the column containing the timestamp data to be plotted on the x-axis. The default value ("detection_timestamp_utc") is is consistent with GLATOS standard.
controlTable	Optional dataframe with two columns, c('location', and 'y_order). The 'location' column is a character vector of locations to be plotted on the y-axis and the name of the 'location' column must match the 'location' colum in detections dataframe (set by detColNames). The 'y_order' column specifies what order the grouping variable will appear on the y-axis (y_order increases as you move away from the x-axis).
plotTitle	An optional character scalar that will apear at the top of the plot. Default is no title.
Ylab	A character scalar indicating the y-axis label that will appear on the figure (default will match detColNames$locationCol).
outFile	An optional character scalar with the name of the png file created (including file extension; default = "AbacusPlot.png").
...	Other plotting arguments that pass to "plot" function (e.g., col, lwd, type).

Details

NAs are not allowed in any of the three required columns of events.

The control table is used to control which locations will appear in the plot and in what order they will appear. If no controlTable is supplied, the function will plot only those locations that appear in the detections data frame and the order of locations on the y-axis will correspond to the order in which each location appears in the data frame.

By default, the function does not distinguish detections from different transmitters and will therefore plot all transmitters the same color. If more than one fish is desired in a single plot, a vector of colors must be passed to the function using the 'col =' argument. The color vector must be the same length as the number of rows in the detections data frame.

#' Alternatively, plots for multiple individual fish can be created by looping through and creating a separate plot on subsetted detections data. Plotting options (i.e., line width and color) can be changed using optional graphical parameters http://www.statmethods.net/advgraphs/parameters.html that are passed to "segments" (see ?segments).

Value

A png file containing the abacus plot (default name "AbacusPlot.png") is written to the working directory.

Author(s)

T. R. Binder

Examples

```
data("walleye_detections") #example data

head(walleye_detections)

#subset one transmitter
walleye_detections <-
 walleye_detections[walleye_detections$transmitter_id == 32123, ]

#plot without control table
abacusPlot(walleye_detections, controlTable=NULL, plotTitle = "TagID: 32123",
 outFile="AbacusPlot_tag32123.png", col = "red")

#get example control table
data("walleye_controlTable") #example dataset
walleye_controlTable

#plot with control table
abacusPlot(walleye_detections, controlTable=walleye_controlTable,
 plotTitle = "TagID: 32123", outFile="AbacusPlot_tag32123_control.png",
 col = "red")

#plot with custom y-axis label and lines connecting symbols
abacusPlot(walleye_detections, controlTable=walleye_controlTable,
 plotTitle = "TagID: 32123", Ylab="Location (GLATOS Array)",
 outFile="AbacusPlot_tag32123_control.png",
 col = "red", type="o")
```

`animatePath` *Create an animated video of spatiotemporal path data*

Description

Create a set of frames (png image files) showing geographic location data (e.g., detections of tagged fish or interpolated path data) at discrete points in time and stitch frames into a video animation (mp4 file).

Usage

```
animatePath(procObj, recs, outDir, background = NULL,
  backgroundYlim = c(41.48, 45.9), backgroundXlim = c(-84, -79.5),
  ffmpeg = NA, plotControl = NULL, procObjColNames = list(animalCol =
  "animal_id", binCol = "bin", timestampCol = "detection_timestamp_utc",
  latitudeCol = "deploy_lat", longitudeCol = "deploy_long", typeCol =
  "record_type"), recColNames = list(latitudeCol = "deploy_lat", longitudeCol
  = "deploy_long", deploy_timestampCol = "deploy_date_time",
  recover_timestampCol = "recover_date_time"))
```

Arguments

procObj	A data frame created by interpolatePath function.
recs	A data frame containing at least four columns with receiver 'lat', 'lon', 'deploy_timestamp', and 'recover_timestamp'. Default column names match GLATOS standard receiver location file (e.g., 'GLATOS_receiverLocations_yyyymmdd.csv'), but column names can also be specified with recColNames.
outDir	A character string with file path to directory where individual frames for animations will be written.
background	An optional object of class SpatialPolygonsDataFrame to be used as background of each frame. Default is a simple polygon of the Great Lakes (greatLakesPoly) included in the 'glatos' package.
backgroundYlim	vector of two values specifying the min/max values for y-scale of plot. Units are same as background argument.
backgroundXlim	vector of two values specifying the min/max values for x-scale of plot. Units are same as background argument.
ffmpeg	A character string with path to install directory for ffmpeg. This argument is only needed if ffmpeg has not been added to your path variable on your computer. For Windows machines, path must point to ffmpeg.exe. For example 'c:\path\to\ffmpeg\bin\ffmpeg.exe'
plotControl	An optional data frame with four columns ('id', 'what', 'color', and 'marker') that specify the plot symbols and colors for each animal and position type. See examples below for an example. • id contains the unique identifer of individual animals and corresponds to 'id' column in 'dtc'. • what indicates if the options should be applied to observed positions (detections; 'detected') or interpolated positions ('interpolated'). • color contains the marker color to be plotted for each animal and position type. • marker contains the marker style to be plotted for each animal and position type.
procObjColNames	A list with names of required columns in procObj:

- `animalCol` is a character string with the name of the column containing the individual animal identifier.
- `binCol` contains timestamps that define each frame.
- `timestampCol` is a character string with the name of the column containing datetime stamps for the detections (MUST be of class 'POSIXct').
- `latitudeCol` is a character string with the name of the column containing latititude of the receiver.
- `longitudeCol` is a character string with the name of the column containing longitude of the receiver.
- `typeCol` is a character string with the name of the optional column that identifies the type of record. Default is 'record_type'.

`recColNames` A list with names of required columns in `recs`:

- `latitudeCol` is a character string with the name of the column containing latitude of the receiver (typically 'deploy_lat' for GLATOS standard detection export data).
- `longitudeCol` is a character string with the name of the column containing longitude of the receiver (typically 'deploy_long' for GLATOS standard detection export data).
- `deploy_timestampCol` is a character string with the name of the column containing datetime stamps for receier deployments (MUST be of class 'POSIXct'; typically 'deploy_date_time'for GLATOS standard detection export data).
- `recover_timestampCol` is a character string with the name of the column containing datetime stamps for receier recover (MUST be of class 'POSIXct'; typically 'recover_date_time'for GLATOS standard detection export data).

Value

Sequentially-numbered png files (one for each frame) and one mp4 file will be written to `outDir`.

Author(s)

Todd Hayden

Examples

```
#example detection data
data(walleye_detections)
head(walleye_detections)

#example receiver location data
data(recLoc_example)
head(recLoc_example)

#call with defaults; linear interpolation
pos1 <- interpolatePath(walleye_detections)

#make sure ffmpeg is installed before calling animatePath
# and if you have not added path to 'ffmpeg.exe' to your Windows PATH
# environment variable then you'll need to do that
# or set path to 'ffmpeg.exe' using the 'ffmpeg' input argument
myDir <- paste0(getwd(),"/frames")
```

```
animatePath(pos1, recs=recLoc_example, outDir=myDir)
```

`calcCollisionProb` *Estimate probability of collision for telemetry transmitters*

Description

Estimate (by simulation) probability of collision for co-located telemetry transmitters with pulse-period-modulation type encoding

Usage

```
calcCollisionProb(delayRng = c(60, 180), burstDur = 5, maxTags = 50,
  nTrans = 10000)
```

Arguments

`delayRng`	A 2-element numeric vector with minimum and maximum delay (time in seconds from end of one coded burst to beginning of next).
`burstDur`	A numeric scalar with duration (in seconds) of each coded burst (i.e., pulse train).
`maxTags`	A numeric scalar with maximum number of co-located transmitters (within detection range at same time).
`nTrans`	A numeric scalar with the number of transmissions to simulate for each co-located transmitter.

Details

Calculates the detection probability associated with collision, given delay range (delayRng), burst duration (burstDur), maximum number of co-located tags (maxTags), and number of simulated transmission per tag (nTrans). The simulation estimates detection probability due only to collisions (i.e., when no other variables influence detection probability) and assuming that all tags are co-located at a receiver for the duration of the simulation.

Value

A data frame containing summary statistics:

`nTags`	Number of tags within detection range at one time
`min`	Minimum detection probability among simulated tags
`q1`	First quartile of detection probabilities among simulated tags
`median`	Median detection probability among simulated tags
`q3`	Third quartile of detection probabilities among simulated tags
`max`	Maximum detection probability among simulated tags
`mean`	Mean detection probability among simulated tags
`expDetsPerHr`	Expected number of detections per hour assuming perfect detection probability, given the number of tags within detection range
`totDetsPerHr`	Observed number of detections per hour for a given number of tags
`effDelay`	Eeffective delay of the transmitter after incorporating collisions
`detsPerTagPerHr`	Mean number of detections per hour per tag

Author(s)

C. Holbrook (cholbrook@usgs.gov) and T. Binder

References

For application example, see:

Binder, T.R., Holbrook, C.M., Hayden, T.A. and Krueger, C.C., 2016. Spatial and temporal variation in positioning probability of acoustic telemetry arrays: fine-scale variability and complex interactions. Animal Biotelemetry, 4(1):1.
http://animalbiotelemetry.biomedcentral.com/articles/10.1186/s40317-016-0097-4

Examples

```
#parameters analagous to Vemco tag, global coding, 45 s nominal delay
foo <- calcCollisionProb(delayRng = c(45, 90), burstDur = 5.12, maxTags = 50,
    nTrans = 10000)
# plot probability of collision by subtracting detection probability from 1
plot(med~nTags, data=foo, type='p', pch=20, ylim=c(0,1),
  xlab="# of transmitters within range", ylab="Probability of collision")
```

---

crw *Simulate a correlated random walk*

Description

Simulate a random walk as series of equal-length steps with turning angles drawn from a normal distribution.

Usage

```
crw(theta = c(0, 5), stepLen = 10, initPos = c(0, 0), initHeading = 0,
  nsteps = 10000)
```

Arguments

`theta`	A 2-element numeric vector with turn angle parameters (theta[1] = mean; theta[2] = sd) from normal distribution.
`stepLen`	A numeric scalar with total distance moved in each step.
`initPos`	A 2-element numeric vector with nital position (initPos[1]=x, initPos[2]=y).
`initHeading`	A numeric scalar with initial heading in degrees.
`nsteps`	A numeric scalar with number of steps to simulate.

Details

First, nsteps turn angles are drawn from a normal distribution. Second, the cumulative sum of the vector of turn angles defines the heading within each step. The x and y component vectors in each are then calculated and summed to obtain the simualted path.

Value

A two-column data frame containing:

x	x coordinates
y	y coordinates

Note

Adapted from code provided by Tom Binder.

Author(s)

C. Holbrook (cholbrook@usgs.gov)

Examples

```
foo <- crw(theta=c(0,5), stepLen=10, initPos=c(0,0), initHeading=0,
  nsteps=10)
plot(foo,type="o",pch=20,asp=c(1,1))
```

`crwInPolygon` *Simulate a correlated random walk inside a polygon*

Description

Uses glatos::crw to simulate a random walk as series of equal-length steps with turning angles drawn from a normal distribution inside a polygon.

Usage

```
crwInPolygon(polyg = data.frame(x = c(0, 0, 150000, 150000), y = c(0, 50000,
  50000, 0)), theta = c(0, 10), stepLen = 100, initPos = c(NA, NA),
  initHeading = NA, nsteps = 30)
```

Arguments

`polyg`	A polygon defined as data frame with columns x and y.
`theta`	A 2-element numeric vector with turn angle parameters (theta[1] = mean; theta[2] = sd) from normal distribution.
`stepLen`	A numeric scalar with total distance moved in each step.
`initPos`	A 2-element numeric vector with nital position (initPos[1]=x, initPos[2]=y).
`initHeading`	A numeric scalar with initial heading in degrees.
`nsteps`	A numeric scalar with number of steps to simulate.

Details

If initPos = NA, then a starting point is randomly selected within the polygon boundary. A path is simulated forward using the crw function. Initial heading is also randomly selected if initHeading = NA. When a step crosses the polygon boundary, a new heading for that step is drawn (from turn angle distribution truncated by polygon boundary), and the remainder of the track is rotated accordingly.

Value

A two-column data frame containing:

x	x coordinates
y	y coordinates

Note

The path is constructed in 1000-step segments (currently hard coded at 1000). In the event that the truncated turn angle distribution (at boundary) leaves no possible solution (i.e., path has entered a crevice in the polygon boundary) then the simulation will stop and an error will indicate that the path is stuck at a boundary. Solution is to try simulation again or to simplify/smooth polygon boundary.

Author(s)

C. Holbrook (cholbrook@usgs.gov)

Examples

```
mypolygon <- data.frame(x=c(-50,-50, 50, 50),y=c(-50,50,50,-50))
foo <- crwInPolygon(mypolygon,theta=c(0,5), stepLen=10, initPos=c(0,0),
  initHeading=0, nsteps=50)
plot(foo,type="o",pch=20,asp=c(1,1),xlim=range(mypolygon$x),
  ylim=range(mypolygon$y))
polygon(mypolygon)
```

---

detectionBubblePlot *Plot number of tagged animals or detections on a map*

Description

Make bubble plots showing the number of fish detected and number of detections across a telemetry receiver network.

Usage

```
detectionBubblePlot(detections, receiverLocs = NULL, map = NULL,
  mapPars = list(xLimits = c(-94.5, -75), yLimits = c(41, 49.5), symbolRadius
  = 1, colGrad = c("white", "red"), showAll = FALSE),
  detColNames = list(locationCol = "glatos_array", animalCol = "animal_id",
  timestampCol = "detection_timestamp_utc", latCol = "deploy_lat", longCol =
  "deploy_long"), recColNames = list(locationCol = "glatos_array", latCol =
  "deploy_lat", longCol = "deploy_long", deploy_timestampCol =
  "deploy_date_time", recover_timestampCol = "recover_date_time"))
```

Arguments

`detections`	A data frame containing detection data with at least 5 columns containing 'location', 'animal', 'timestamp', 'latitude', and 'longitude' data. Default column names match GLATOS standard detection export file (e.g., '_detectionsWithLocs.csv'), but column names can also be specified with `detColNames`.
`receiverLocs`	An optional data frame containing at least 5 columns with receiver 'location', 'lat', 'lon', 'deploy_timestamp', and 'recover_timestamp'. Default column names match GLATOS standard receiver location file (e.g., 'GLATOS_receiverLocations_yyyymmdd.csv'), but column names can also be specified with `recColNames`.
`map`	An optional SpatialPolygonsDataFrame or other geo-referenced object to be plotted as the background for the plot. The default is a SpatialPolygonsDataFrame of the Great Lakes (e.g., `data(greatLakesPoly)`.
`mapPars`	A list of optional mapping parameters (with exact names matching below) including: • `xLimits` is a two-element numeric vector that defines minimum and maximum extents of the viewable plot area along the x-axis. • `yLimits` is a two-element numeric vector that defines minimum and maximum extents of the viewable plot area along the y-axis. • `symbolRadius` is a numeric scalar that sets the radius of each "bubble" on the plot in units of percent of x-axis scale. Default value = 1 (i.e., 1 • `colGrad` A two-element character vector indicating the start and end colors of the gradient scale used to color-code "bubbles". • `showAll` A logical (default = FALSE) indicating whether to plot all receiver groups (TRUE) or only those receiver groups on which the fish were detected (FALSE).
`detColNames`	A list with names of required columns in `detections`: • `locationCol` is a character string with the name of the column containing the locations that will be plotted (typically 'glatos_array' or 'station' for GLATOS standard detection export data). • `animalCol` is a character string with the name of the column containing the individual animal identifier (typically 'transmitter_id' or 'animal_id' for GLATOS standard detection export data). • `timestampCol` is a character string with the name of the column containing datetime stamps for detections (MUST be of class 'POSIXct'; typically 'detection_timestamp_utc'for GLATOS standard detection export data). • `latitudeCol` is a character string with the name of the column containing latitude of the receiver (typically 'deploy_lat' for GLATOS standard detection export data). • `longitudeCol` is a character string with the name of the column containing longitude of the receiver (typically 'deploy_lat' for GLATOS standard detection export data).
`recColNames`	A list with names of required columns in `receiverLocs`: • `locationCol` is a character string with the name of the column containing the locations that will be plotted (typically 'glatos_array' or 'station' for GLATOS standard detection export data). • `latitudeCol` is a character string with the name of the column containing latitude of the receiver (typically 'deploy_lat' for GLATOS standard detection export data).

- `longitudeCol` is a character string with the name of the column containing longitude of the receiver (typically 'deploy_long' for GLATOS standard detection export data).
- `deploy_timestampCol` is a character string with the name of the column containing datetime stamps for receier deployments (MUST be of class 'POSIXct'; typically 'deploy_date_time'for GLATOS standard detection export data).
- `recover_timestampCol` is a character string with the name of the column containing datetime stamps for receier recover (MUST be of class 'POSIXct'; typically 'recover_date_time'for GLATOS standard detection export data).

Details

If `mapPars$showAll` is TRUE then the plot will show all receivers, including those that detected none of the transmitters in `detections`. In that case, it will first be determined if a receiver (or group of receivers, as defined by 'location') was in the water during the time interval between the first and last detection in `detections` data frame. Receivers for which deployment-recovery intervals do not with the time coverage of detections will not be included in the plot.

"ColGrad" is used in a call to `colorRampPalette()`, which will accept a vector containing any two colors return by `colors()` as character strings.

Value

A list containing the following data frames and columns:

`summaryNumFish`
- `location`: user-specified 'location' labels
- `Summary`: number of unique 'animals' detected at 'location'
- `meanLat`: mean latitude of receivers at 'location'
- `meanLon`: mean longitude of receivers at 'location'

`summaryNumDetections`
- `location`: user-specified 'location' labels
- `Summary`: number of unique detections at 'location'
- `meanLat`: mean latitude of receivers at 'location'
- `meanLon`: mean longitude of receivers at 'location'

Two png files containing bubble plots for number of unique fish detected ("BubblePlot_summaryNumFish.png") and total detections ("BubblePlot_summaryNumDetections.png") are also written to the working directory. Summary data for each plot are also written to CSV files in the working directory.

Author(s)

T. R. Binder

Examples

```
#example detection data
data(walleye_detections)
head(walleye_detections)

#call with defaults
detectionBubblePlot(walleye_detections)
```

```
#example receiver location data
data(recLoc_example)
head(recLoc_example)

#view example map background
library(sp) #to avoid errors plotting SpatialPolygonsDataFrame
data(greatLakesPoly)
plot(greatLakesPoly)

detectionBubblePlot(walleye_detections, receiverLocs=recLoc_example,
  mapParms=list(symbolRadius = 1.4,colGrad = c("white", "blue"), showAll=T))
```

`detectionEventFilter` *Classify discrete events in detection data*

Description

Reduce detection data from an acoustic telemetry receiver into discrete detection events, defined by movement between receivers (or receiver groups, depending on location), or sequential detections at the same location that are separated by a user-defined threshold period of time.

Usage

```
detectionEventFilter(detections, detColNames = list(locationCol =
  "glatos_array", animalCol = "animal_id", timestampCol =
  "detection_timestamp_utc", latCol = "deploy_lat", longCol = "deploy_long"),
  timeSep = Inf)
```

Arguments

`detections`	A data frame containing detection data with at least 5 columns containing 'location', 'animal', 'timestamp', 'latitude', and 'longitude'. Column names are specified with `detColNames`.
`detColNames`	A list with names of required columns in `detections`: • `locationCol` is a character string with the name of the column containing locations you wish to filter to (typically 'glatos_array' or 'station' for GLATOS data). • `animalCol` is a character string with the name of the column containing the individual animal identifier. • `timestampCol` is a character string with the name of the column containing datetime stamps for the detections (MUST be of class 'POSIXct'). • `latitudeCol` is a character string with the name of the column containing latititude of the receiver. • `longitudeCol` is a character string with the name of the column containing longititude of the receiver.
`timeSep`	Amount of time (in seconds) that must pass between sequential detections on the same receiver (or group of receivers, depending on specified location) before that detection is considered to belong to a new detection event. The default value `Inf`, will not define events based on elapsed time (only when location changes).

Details

MeanLatitude and MeanLongitude columns in the output dataframe are the mean GPS locations for the detections comprising that detection event. For example, if the a fish was detected at 3 receiver stations in a glatos_array and glatos_array was selected as the location, then GPS location for that event will be the mean of the latitude and longitude for those three receiver stations (weighted based on the number of detections that occurred on each station).

Value

A data frame containing discrete detection event data:

`MeanLatitude`	Mean latitude of detections comprising each event.
`MeanLongitude`	Mean longitude of detections comprising each event.
`FirstDetection`	The time of the first detection in a given detection event.
`LastDetection`	The time of the last detection in a given detection event.
`NumDetections`	The total number of detection that comprised a given detection event.
`ResTime_sec`	The elapsed time in seconds between the first and last detection in a given event.

Author(s)

T. R. Binder

Examples

```
library(glatos)
data("walleye_detections") #example data

head(walleye_detections)

filt0 <- detectionEventFilter(walleye_detections) #no time filter

#7-day filter
filt_7d <- detectionEventFilter(walleye_detections , timeSep = 604800)
```

---

detectTransmissions *Simulate detection of transmitter signals in a receiver network*

Description

Simulates detection of transmitter signals in a receiver network based on detection range curve (detection probability as a function of distance), location of transmitter, and location of receivers.

Usage

```
detectTransmissions(trnsLoc = NA, recLoc = NA, detRngFun = NA)
```

Arguments

trnsLoc	A three-column data frame with locations (numeric columns named 'x' and 'y') and timestamps (numeric or POSIXct column named 'et') where signals were transmitted.
recLoc	A two-column data frame with receiver locations (numeric columns named 'x' and 'y')
detRngFun	A function that defines detection range curve; must accept a numeric vector of distances and return a numeric vector of detection probabilities at each distance.

Details

Distances between each signal transmission location and receiver are calculated using pythagorean theorem. The probability of detecting each signal on each receiver is determined from the detection range curve. Detection of each signal on each receiver is determined stochastically by draws from a Bernoulli distribution with probability p (detection prob).

This function was written to be used along with transmitAlongPath.

Value

A data frame containing:

trns_id	Unique signal transmission ID
recv_id	Unique receiver ID
recv_x	Receiver x coordinate
recv_y	Receiver y coordinate
trns_x	Transmitter x coordinate
trns_y	Transmitter y coordinate
etime	Elapsed time

Author(s)

C. Holbrook (cholbrook@usgs.gov)

See Also

transmitAlongPath to simulate transmissions along a path (i.e., create trnsLoc).

Examples

```
#make a simple path in polygon
mypath <- crwInPolygon(data.frame(x = c(0, 0, 1000, 1000),
  y = c(0, 1000, 1000, 0)), stepLen=100, nsteps=50)
plot(mypath,type='l',xlim=c(0,1000),ylim=c(0,1000)) #view path

#add receivers
recs <- expand.grid(c(250,750),c(250,750))
names(recs) <- c("x","y") #needed by detectTransmissions
points(recs, pch=15, col="blue")

#simulate tag transmissions
mytrns <- transmitAlongPath(mypath,vel=2.0,delayRng=c(60,180),burstDur=5.0)
points(mytrns,pch=21) #add to plot
```

```
#Define detection range function (to pass as detRngFun)
# that returns detection probability for given distance
# assume logistic form of detection range curve where
#   dm = distance in meters
#   b = intercept and slope
pdrf <- function(dm, b=c(0.5, -1/120)){
  p <- 1/(1+exp(-(b[1]+b[2]*dm)))
  return(p)
}
pdrf(c(100,200,300,400,500)) #view detection probs. at some distances

#simulate detection
mydtc <- detectTransmissions(trnsLoc=mytrns, recLoc=recs, detRngFun=pdrf)
#view transmissions that were detected
points(trns_y~trns_x, data=mydtc,pch=21, bg="red")
```

`eventPlot` *Plot time series event data*

Description

Create an abacus-like plot for discrete event data with start and end times. Suitable for data like detection events (i.e., output from the glatos package `glatos::detectionEventFilter` function) and receiver histories (i.e., based on receiverLocations.csv GLATOS export).

Usage

```
eventPlot(events, eventColNames = list(locationCol = "Location", eventStartCol
  = "FirstDetection", eventEndCol = "LastDetection"), controlTable = NULL,
  plotTitle = "", Ylab = NA, outFile = "EventPlot.png", ...)
```

Arguments

`events`	A data frame containing at least three columns with 'location', 'eventStart', and 'eventEnd' data. Default column names match those produced by `glatos::detectionEventFilter`, but non-standard column names can be specified with `eventsColNames`.
`eventColNames`	An optional list of character strings with names of required columns in `events`. • `locationCol` is a character string with the name of the column containing the locations that will be plotted on the y-axis. • `eventStartCol` is is a character string with the name of the column containing the date-time stamps for the start of each event. Must be of class POSIXct. • `eventEndCol` is is a character string with the name of the column containing the date-time stamps for the end of each event. Must be of class POSIXct.
`controlTable`	An optional dataframe with two columns: 'location' and 'y_order'. The 'location' column is a character vector of locations to be plotted on the y-axis and the name of the 'location' column must match the 'location' colum in `detections`

	dataframe (set by `detColNames`). The 'y_order' column specifies what order the grouping variable will appear on the y-axis (y_order increases as you move away from the x-axis).
`plotTitle`	An optional character scalar that will apear at the top of the plot. Default is no title.
`Ylab`	A character scalar indicating the y-axis label that will appear on the figure (default will match `eventColNames$locationCol`).
`outFile`	An optional character scalar with the name of the png file created (including file extension; default = "eventPlot.png").
`...`	Other plotting arguments that pass to "plot" function (e.g., col, lwd, type).

Details

NAs are not allowed in any of the three required columns of `events`.

The control table is used to control which locations will appear in the plot and in what order they will appear. If no controlTable is supplied, the function will plot only those locations that appear in the `events` data frame and the order of locations on the y-axis will correspond to the order in which each location appears in the data frame.

By default, the function does not distinguish detections from different transmitters and will therefore plot all transmitters the same color. If more than one fish is desired in a single plot, a vector of colors must be passed to the function using the 'col =' argument. The color vector must be the same length as the number of rows in the events data frame.

#' Alternatively, plots for multiple individual fish can be created by looping through and creating a separate plot on subsetted events data. Plotting options (i.e., line width and color) can be changed using optional graphical parameters `http://www.statmethods.net/advgraphs/parameters.html` that are passed to "segments" (see ?segments).

Value

A png file containing the events plot (default name "EventPlot.png") is written to the working directory.

Author(s)

T. R. Binder

Examples

```
#make detection event file using detectionEventFilter function
data(walleye_detections)
#subset one transmitter
walleye_detections <-
 walleye_detections[walleye_detections$transmitter_id == 32123, ]

#make events based on 7-d time threshold
walleye_events_7d <- detectionEventFilter(walleye_detections,
 timeSep = 604800)

#plot with defaults
eventPlot(walleye_events_7d)

#example control table
```

```
data(walleye_controlTable)
head(walleye_controlTable)
names(walleye_controlTable)[1] <- "Location" #to match events

#plot with controlTable and customized axes
eventPlot(walleye_events_7d, controlTable = walleye_controlTable,
  plotTitle = "Receiver History", Ylab = "Location", col = "red", lwd = 3)
```

`falseDetectionFilter` *False detection filter*

Description

Identify possible false detections based on "short interval" criteria

Usage

```
falseDetectionFilter(detections, tf, minLagCol = "min_lag")
```

Arguments

`detections`	A data frame containing detection data (e.g., from the standard GLATOS detection export file '*_detectionsWithLocs.csv'). Must contain the column `min_lag` or equivalent (specified by `minLagCol`). `min_lag` is loosely based on the the "short interval" method described by Pincock (2012). In this case (GLATOS), it is defined for each detection as the shortest interval (in seconds) between either the previous or next detection (whichever is closest) of the same transmitter on the same receiver.
`tf`	A numeric scalar indicating the time threshold (in seconds; e.g., Pincock's (2012) "short interval") for identifying possible false detections.
`minLagCol`	A character string containing the name of the column in `detections` that contains 'min_lag'.

Details

Detections are identified as potentially false when `min_lag > tf`.

A new column (`passedFilter`), indicating if each record (row) passed the filter, is added to the input data frame. This function was written specifically with GLATOS standard detection export in mind, so it requires `min_lag` or equivalent.

A common rule of thumb for choosing tf for VEMCO PPM encoded transmitters is 30 times the nominal delay (e.g., 3600 s for a transmitter with a 120 s nominal delay) - see Pincock (2012).

Value

A dataframe consisting of `detections` with an additional column 'passedFilted' indicating if each detection did (1) or did not (0) pass the criteria.

Author(s)

T. R. Binder

References

Pincock, D.G., 2012. False detections: what they are and how to remove them from detection data. Vemco Division, Amirix Systems Inc., Halifax, Nova Scotia.
http://www.vemco.com/pdf/false_detections.pdf

Simpfendorfer, C.A., Huveneers, C., Steckenreuter, A., Tattersall, K., Hoenner, X., Harcourt, R. and Heupel, M.R., 2015. Ghosts in the data: false detections in VEMCO pulse position modulation acoustic telemetry monitoring equipment. Animal Biotelemetry, 3(1), p.55.
https://animalbiotelemetry.biomedcentral.com/articles/10.1186/s40317-015-0094-z

Examples

```
data("walleye_detections") #example data

head(walleye_detections)

dtx <- falseDetectionFilter(walleye_detections, 3600)
head(dtx)
```

`glatos` *An R package for the Great Lakes Acoustic Telemetry Observation System*

Description

`glatos` is an R package with functions useful to members of the Great Lakes Acoustic Telemetry Observation System (http://glatos.glos.us). Functions may be generally useful for processing, analysing, simulating, and visualizing acoustic telemetry data, but are not strictly limited to acoustic telemetry applications.

Package status

This package is in early development. If you encounter problems or have questions or suggestions, please post a new issue. If you have code to contribute, feel free to start a new branch. Any questions or comments can be sent to cholbrook@usgs.gov (maintainer: Chris Holbrook).

Simulation functions for system design and evaluation

clacCollisionProb estimates the probability of collisions for PPM-type co-located telemetry transmitters. This is useful for determining the number of fish to release or tag specifications (e.g., delay).

receiverLineDetSim simulates detection of acoustic-tagged fish crossing a receiver line (or single receiver). This is useful for determining optimal spacing of receviers in a line and tag specifications (e.g., delay).

crwInPolygonR, transmitAlongPath, and detectTransmissions individually simulate random fish movement paths within a water body (crwInPolygonR: a random walk in a polygon), tag signal transmissions along those paths (transmitAlongPath: time series and locations of transmissions based on tag specs), and detection of those transmittions by receivers in a user-defined receiver network (detectTransmissions: time series and locations of detections based on detection range curve). Collectively, these functions can be used to explore, compare, and contrast theoretical performance of a wide range of transmitter and receiver network designs.

Data processing and summarization

falseDetectionFilter identifies potential false detections in the GLATOS standard data export package using "short interval" criteria (GLATOS min_lag column).

detectionEventFilter distills detection data down to a much smaller number of discrete detection events, defined as a change in location (defined by user) or time gap that exceeds a threshold (defined by user).

Visualization and data exploration

kmlWorkbook is useful for exploring receiver and animal release locations in Google Earth.

abacusPlot and detectionEventPlot are useful for exploring movement patterns of individual tagged animals.

detectionBubblePlot is useful for exploring distribution of tagged individuals among receivers.

movePath,interpolatePath,andanimatePath can be used together to interpolate movement paths between detections and save animated movement paths to a video file (mp4).

Random Utility functions

vrl2csv converts a Vemco VRL file to a comma separated values (CSV) file using a system call to VEMCO VUE convert command.

rotatePoints will rotate a set of 2-d points about another point.

crw will simulate an unconstrained correlated random walk.

vectorHeading will calculate (in degrees) the heading of the vector between adjacent point-pairs in a set of positions (e.g., along a track).

Installation

The R package GLATOS is availabed from the Ocean Tracking Network's gitlab `https://gitlab.oceantrack.org/chrisholbrook/glatos`.

To install:

1. install devtoools for R (if you haven't already)
```
> install.packages("devtools")
```

2.replace USERNAME and PASSWORD in the R code below with your own
```
> library(devtools)
```

3. download the package and install
```
> install_git("https://USERNAME:PASSWORD@gitlab.oceantrack.org/chrisholbrook/glatos.git")
```

`interpolatePath` *Interpolate new positions within a spatiotemporal path data*

Description

Interpolate new positions within a spatiotemporal path data set (e.g., detections of tagged fish) at regularly-spaced time intervals using linear or non-linear interpolation (via `movePath`).

Usage

```
interpolatePath(dtc, intTimeStamp = 86400, rast = NULL, lnlThresh = 0.9,
  detColNames = list(individualCol = "animal_id", timestampCol =
  "detection_timestamp_utc", latitudeCol = "deploy_lat", longitudeCol =
  "deploy_long", typeCol = "record_type"))
```

Arguments

`dtc`	A data frame containing spatiotemporal data with at least 4 columns containing 'individual', 'timestamp','longitude', and 'latitude' data and an optional fifth column with the 'type' of record (e.g., detection). Default column names match the GLATOS detection export file but other names can be specified with `detColNames`.
`intTimeStamp`	The time step size (in seconds) of interpolated positions. Default is 86400 (one day).
`rast`	An optional transition matrix with the "cost" of moving across each cell within the map extent. Must be of class `TransitionLayer` (See `gdistance` package). Passed to `trans` in `movePath`.
`lnlThresh`	A numeric threshold for determining if linear or non-linear interpolation will be used based on the ratio of linear-to-non-linear shortest path distances. Passed to `ithresh` in `movePath`.
`detColNames`	A list with names of columns in `dtc`: • `individualCol` is a character string that uniquely identies an an indvidual (e.g., tagged animal). Default is 'animal_id'. • `timestampCol` is a character string with the name of the column containing datetime stamps. Default is 'detection_timestamp_utc'. • `latitudeCol` is a character string with the name of the column containing latitude data. Default is 'deploy_lat'. • `longitudeCol` is a character string with the name of the column containing longitude of the receiver. Default is 'deploy_long'. • `typeCol` is a character string with the name of the optional column that identifies the type of record. Default is 'record_type'.

Details

Interpolation is done by passing each consecutive pair of points to `movePath` for interpolation via linear or non-linear methods, depending on `rast`.

Non-linear interpolation uses the 'gdistance' package to find the shortest pathway between two locations (i.e., receivers) that avoid 'impossible' movements (e.g., over land for fish). The shortest non-linear path between two locations is calculated using a 'transition matrix layer' (`rast`) that represents the 'cost' of an animal moving between adjacent grid cells. For example, each cell in `rast` may be coded as water (1) or land (0) to represent possible (1) and impossible(0) movement path.

Linear interpolation is used for all points when `rast` is not supplied. When `rast` is supplied, then interpolation method is determined for pair of observed positions. For example, linear interpolaton will be used if the two points are exactly the same and when the ratio of linear-to-non- can be used to control whether non-linear or linear interpolation is used for all points. For example, non-linear interpolation will be used for all points when `lnlThresh` = 1 and linear interpolation will be used for all points when `lnlThresh` = 0.

Value

A dataframe with id, timestamp, lat, lon, and record type.

Author(s)

Todd Hayden

See Also

movePath

Examples

```
------------------------------------------------
#EXAMPLE #1 - simple example

#example transition matrix
data(greatLakesTrLayer)

#example map background
data(greatLakesPoly)
library(sp) #to plot SpatialPolygon without error
plot(greatLakesPoly)

#make up points points
pos <- data.frame(
  id=1,
  x=c(-87,-82.5, -78),
  y=c(44, 44.5, 43.5),
  time=as.POSIXct(c("2000-01-01 00:00",
    "2000-02-01 00:00", "2000-03-01 00:00")))

#coerce to SpatialPoints object and plot
pts <- SpatialPoints(pos[,c("x","y")])
points(pts, pch=20, col='red', cex=3)

#interpolate path using linear method
path1 <- interpolatePath(pos,
  detColNames=list(individualCol="id", timestampCol="time",
 longitudeCol="x", latitudeCol="y"))

#coerce to SpatialPoints object and plot
pts1 <- SpatialPoints(path1[,c("x","y")])
points(pts1, pch=20, col='blue', lwd=2, cex=1.5)

#example transition matrix
data(greatLakesTrLayer)

#interpolate path using non-linear method (requires 'trans')
path2 <- interpolatePath(pos,
  rast=greatLakesTrLayer,
  detColNames=list(individualCol="id", timestampCol="time",
  longitudeCol="x", latitudeCol="y"))

#coerce to SpatialPoints object and plot
```

```
pts2 <- SpatialPoints(path2[,c("x","y")])
points(pts2, pch=20, col='green', lwd=2, cex=1.5)

#can also force linear-interpolation with lnlThresh=0
path3 <- interpolatePath(pos,
  rast=greatLakesTrLayer, lnlThresh=0,
  detColNames=list(individualCol="id", timestampCol="time",
  longitudeCol="x", latitudeCol="y"))

#coerce to SpatialPoints object and plot
pts3 <- SpatialPoints(path3[,c("x","y")])
points(pts3, pch=20, col='magenta', lwd=2, cex=1.5)

-------------------------------------------------
#EXAMPLE #2 - GLATOS detection data
data(walleye_detections)
head(walleye_detections)

#call with defaults; linear interpolation
pos1 <- interpolatePath(walleye_detections)

#plot on example map background
data(greatLakesPoly)
library(sp) #to plot SpatialPolygon without error
plot(greatLakesPoly)

#coerce to SpatialPoints object and plot
pts1 <- SpatialPoints(pos1[pos1$animal_id==3,c("deploy_long","deploy_lat")])
points(pts1, pch=20, col='red', cex=0.5)

#example transition matrix
data(greatLakesTrLayer)

#call with "transition matrix" (non-linear interpolation), other options
# note that it is quite a bit slower due than linear interpolation
pos2 <- interpolatePath(walleye_detections, rast=greatLakesTrLayer)

#coerce to SpatialPoints object and plot
pts2 <- SpatialPoints(pos2[,c("deploy_long","deploy_lat")])
points(pts2, pch=20, col='blue', cex=0.5)
```

---

kmlWorkbook *Make a KML (or KMZ) file for viewing receiver and animal release locations in Google Earth.*

Description

Receiver data (deployment location, deployment timestamp, and recovery timestamp) and tagging data (release location, release timestamp) are imported from a zipped GLATOS Workbook archive and used to make a KML (and optionally, KMZ) for viewing receiver deployments and release locations in Google Earth.

Usage

```
kmlWorkbook(zipFile, browse = F, kmz = F, labelSize = 0.6,
  iconSize = 0.6, showOngoing = T, endDate = "2020-01-01")
```

Arguments

browse	A logical scalar. If TRUE, user is asked to select zipfile using windows explorer. Default value is FALSE.
kmz	A logical scalar; If TRUE, a KMZ file (zipped KML file) will also be created. Default value is FALSE.
labelSize	A numeric scalar with the size of placemark labels (only shown when placemark is highlighted by user).
iconSize	A numeric scalar with the size of placemark icons.
showOngoing	A logicalscalar that indicates if ongoing stations (missing recovery timestamp) should be included in result.
endDate	End date (e.g. "YYYY-MM-DD") to be used for any ongoing stations (if showOngoing == T)
zipfile	A character vector with the full path and filename of zipped GLATOS workbook (this is the **ZIPPED** archive that gets uploaded to GLATOSWeb)

Details

Receiver locations will be visible between deployment and recovery timestamps at each location. Release locations will be displayed when the display window includes the date of release.

Value

A KML (and optionally, KMZ) file, written to the directory that contains the zipped GLATOS workbook. Nothing is returned to the R console.

Author(s)

C. Holbrook (cholbrook@usgs.gov)

Examples

```
#get path to example GLATOS Data Workbook
zipFile <- system.file("extdata",
  "SMRSL_GLATOS_20140828.xlsm.zip",package="glatos")
kmlWorkbook(zipFile,browse=F,kmz=T,labelSize=0.6,iconSize=0.6,
  showOngoing=T,endDate="2020-01-01")
```

`movePath` *Interpolate positions within a pair of geographic coordinates using using linear ("great circle") or non-linear ("lowest cost") methods*

Description

`movePath` calculates interpolated positions between a pair of coordinates using linear ("great circle") or non-linear ("lowest cost") interpolation methods. The main purpose of this function is to prevent interpolation of movement paths through inaccessible locations (e.g., over land for fish).

Usage

```
movePath(startX, startY, endX, endY, iTime, trans = NULL, iThresh = 0.9)
```

Arguments

`startX`	start longitude, vector of length 1
`startY`	start latitude, vector of length 1
`endX`	end longitude, vector of length 1
`endY`	end latitude, vector of length 1
`iTime`	A vector of datetime stamps (MUST be of class POSIXct) at which to interpolate geographic positions between start and end locations. The first element must correspond with the position at `startX` and `startY` and the last element must correspond with the position at `startX` and `startY`.
`trans`	An optional transition matrix with the "cost" of moving across across each cell within the map extent. Must be of class `TransitionLayer`. See `gdistance` package.
`iThresh`	A numeric threshold for determining if linear or non-linear interpolation will be used based on the ratio of linear-to-non-linear shortest path distances.

Details

This function operates on a single pair of geographic (start and end) positions and is the primary function in `interpolatePath`.

Non-linear interpolation uses the 'gdistance' package to find the shortest pathway between two locations (i.e., receivers) that avoid 'impossible' movements (e.g., over land for fish). The shortest non-linear path between two locations is calculated using a 'transition matrix layer' (`trans`) that represents the 'cost' of an animal moving between adjacent grid cells. For example, each cell in `trans` may be coded as water (1) or land (0) to represent possible (1) and impossible(0) movement path.

If `trans` is not specified, or if the start and end positions are the same, then new positions are calculated using linear interpolation according to the values in `iTime`. If `trans` is specified and start and end points are different, then method used (linear or non-linear) depends on the ratio of linear-to-non-linear distances between points. Linear interpolation will be used when the ratio is larger than `iThresh` and non-linear interpretation will be used when the ratio is smaller than `iThresh`. `iThresh` can be used to control whether non-linear or linear interpolation is used for all points. For example, non-linear interpolation will be used for all points when `iThresh` = 1 and linear interpolation will be used for all points when `iThresh` = 0.

Value

Dataframe with interpolated timestamp, lat, and lon

Author(s)

Todd Hayden

See Also

interpolatePath

Examples

```
#example transition matrix
data(greatLakesTrLayer)

#example map background
data(greatLakesPoly)
library(sp) #to plot SpatialPolygon without error
plot(greatLakesPoly)

#make up two points
x <- c(-87,-82.5)
y <- c(44, 44.5)

#coerce to SpatialPoints object and plot
pts <- SpatialPoints(cbind(x,y))
points(pts, pch=20, col='red', cex=3)

#interpolate path using linear method
path1 <- movePath(startX=x[1], startY=y[1], endX=x[2], endY=y[2],
  iTime=as.POSIXct("2000-01-01 00:00") + 1:30)

#coerce to SpatialPoints object and plot
pts1 <- SpatialPoints(path1[,c("lon","lat")])
points(pts1, pch=20, col='blue', lwd=2, cex=1.5)

#interpolate path using non-linear method (requires 'trans')
path2 <- movePath(startX=x[1], startY=y[1], endX=x[2], endY=y[2],
  iTime=as.POSIXct("2000-01-01 00:00") + 1:30,
  trans=greatLakesTrLayer)

#coerce to SpatialPoints object and plot
pts2 <- SpatialPoints(path2[,c("lon","lat")])
points(pts2, pch=20, col='green', lwd=2, cex=1.5)

#can also force linear interpolation by setting 'lnlThresh' = 0
path3 <- interpolatePath(pos,
  rast=greatLakesTrLayer, lnlThresh=0,
  detColNames=list(individualCol="id", timestampCol="time",
  longitudeCol="x", latitudeCol="y"))

pts3 <- SpatialPoints(path3[,c("x","y")])
points(pts3, pch=20, col='magenta', lwd=2, cex=1.5)
```

`receiverLineDetSim` *Simulate detection of acoustic-tagged fish crossing a receiver line*

Description

Estimate, by simulation, the probability of detecting an acoustic-tagged fish on a receiver line, given constant fish velocity (ground speed), receiver spacing, number of receivers, and detection range curve.

Usage

```
receiverLineDetSim(vel = 1, delayRng = c(120, 360), burstDur = 5,
  recSpc = 1000, maxDist = 2000, rngFun, outerLim = c(0, 0),
  nsim = 1000, showPlot = FALSE)
```

Arguments

`vel`	A numeric scalar with fish velocity in meters per second.
`delayRng`	A 2-element numeric vector with minimum and maximum delay (time in seconds from end of one coded burst to beginning of next)
`burstDur`	A numeric scalar with duration (in seconds) of each coded burst (i.e., pulse train).
`recSpc`	A numeric vector with distances (in meters) between receivers. The length of vector is N-1, where N is number of receivers. One receiver is simulated when `recSpc = NA` (default).
`maxDist`	A numeric scalar with maximum distance between tagged fish and any receiver during simulation (i.e., sets spatial boundaries)
`rngFun`	A function that defines detection range curve; must accept a numeric vector of distances and return a numeric vector of detection probabilities at each distance.
`outerLim`	A two-element numeric vector with space (in meters) in which simulated fish are allowed to pass to left (first element) and right (second element) of the receiver line.
`nsim`	Integer scalar with the number of crossings (fish) to simulate
`showPlot`	A logical scalar. Should a plot be drawn showing receivers and fish paths?

Details

Virtual tagged fish (N=`nsim`) are "swum" through a virtual receiver line. The first element of `recSpc` determines spacing between first two receivers in the line, and each subsequent element of `recSpc` determine spacing of subsequent receivers along the line, such that the number of receivers is equal to `length(recSpc) + 1`. Each fish moves at constant velocity (`vel`) along a line perpendicular to the receiver line. The location of each fish path along the receiver line is random (drawn from uniform distribution), and fish can pass outside the receiver line (to the left of the first receiver or right of last receiver) if `outerLim[1]` or `outerLim[2]` are greater than 0 meters. Each fish starts and ends about `maxDist` meters from the receiver line.

A simulated tag signal is transmitted every `delayRng[1]` to `delayRng[2]` seconds. At time of each transmission, the distance is calculated between the tag and each receiver, and rngFun is used to calculate the probability (p) that the signal was detected on each receiver. Detection or non-detection on each receiver is determined by a draw from a Bernoulli distribution with probability p.

Value

A data frame with one column:

detProb	The proportion of simulated fish that were detected more than once on any single receiver.

Author(s)

C. Holbrook (cholbrook@usgs.gov)

References

For application example, see:

Hayden, T.A., Holbrook, C.M., Binder, T.R., Dettmers, J.M., Cooke, S.J., Vandergoot, C.S. and Krueger, C.C., 2016. Probability of acoustic transmitter detections by receiver lines in Lake Huron: results of multi-year field tests and simulations. Animal Biotelemetry, 4(1), p.19.
https://animalbiotelemetry.biomedcentral.com/articles/10.1186/s40317-016-0112-9

Examples

```
#EXAMPLE 1 - simulate detection on line of ten receivers

 #Define detection range function (to pass as rngFun)
 # that returns detection probability for given distance
  # assume logistic form of detection range curve where
  #   dm = distance in meters
  #   b = intercept and slope
 pdrf <- function(dm, b=c(5.5, -1/120)){
    p <- 1/(1+exp(-(b[1]+b[2]*dm)))
    return(p)
 }

 #preview detection range curve
 plot(pdrf(0:2000),type="l",ylab="Probability of detecting each coded burst",
xlab="Distance between receiver and transmitter")

 #Simulate detection using pdrf; default values otherwise
 dp <- receiverLineDetSim(rngFun=pdrf)
 dp

 #Again with only 10 virtual fish and optional plot to see simulated data
 dp <- receiverLineDetSim(rngFun=pdrf, nsim=10, showPlot=T) #w/ optional plot
 dp

 #Again but six receivers and allow fish to pass to left and right of line
 dp <- receiverLineDetSim(rngFun=pdrf, recSpc=rep(1000,5),
outerLim=c(1000, 1000), nsim=10, showPlot=T)
 dp

 #Again but four receivers with irregular spacing
 dp <- receiverLineDetSim(rngFun=pdrf, recSpc=c(2000,4000,2000),
  outerLim=c(1000, 1000), nsim=10, showPlot=T)
 dp
```

```
#EXAMPLE 2 - summarize detection probability vs. receiver spacing

 #two receivers only, spaced 'spc' m apart
 #define scenarious where two receiver are spaced
 spc <- seq(100,5000, 100) #two receivers spaced 100, 200, ... 5000 m
 #loop through scenarious, estimate detection probability for each
 for(i in 1:length(spc)){
   if(i==1) dp <- numeric(length(spc)) #preallocate
   dp[i] <- receiverLineDetSim(recSpc=spc[i], rngFun=pdrf)
 }
 cbind(spc,dp) #view results
 #plot results
 plot(spc, dp, type="o",ylim=c(0,1),
   xlab="distance between receivers in meters",
  ylab="proportion of virtual fish detected")
 # e.g., >95% virtual fish detected up to 1400 m spacing in this example


#EXAMPLE 3 - summarize detection probability vs. fish swim speed

 #define scenarios of fish movement rate
 swim <- seq(0.1, 5.0, 0.1) #constant velocity
 for(i in 1:length(swim)){
   if(i==1) dp <- numeric(length(swim)) #preallocate
   dp[i] <- receiverLineDetSim(vel=swim[i], rngFun=pdrf)
 }
 cbind(swim,dp) #view results
 #plot results
 plot(swim, dp, type="o", ylim=c(0,1), xlab="fish movement rate, m/s",
  ylab="proportion of virtual fish detected")
 # e.g., >95% virtual fish detected up to 1.7 m/s rate in this example
 # e.g., declines linearly above 1.7 m/s


#EXAMPLE 4 - empirical detection range curve instead of logistic

 #create data frame with observed det. efficiency (p) at each distance (x)
 edr <- data.frame(
   x=c(0,363,444,530,636,714,794,889,920), #tag-receiver distance
   p=c(1,1,0.96,0.71,0.67,0.75,0.88,0.21,0)) # detection prob

 #now create a function to return the detection probability
 # based on distance and linear interpolation within edr
 # i.e., estimate p at given x by "connecting the dots"
 edrf <- function(dm, my.edr=edr) {
   p <- approx(x=my.edr$x,y=my.edr$p,xout=dm, rule=2)$y
   return(p)
 }

 #preview empirical detection range curve
 plot(edrf(0:2000),type="l",
   ylab="probability of detecting each coded burst",
  xlab="distance between receiver and transmitter, meters")

 #use empirical curve (edrf) in simulation
 dp <- receiverLineDetSim(rngFun=edrf, nsim=10, showPlot=T) #w/ optional plot
 dp
```

`rotatePoints` *Rotate points in a 2-d plane*

Description

Rotate points around a point in a 2-d plane

Usage

```
rotatePoints(x, y, theta, focus)
```

Arguments

`x`	A numeric vector of x coordinates; minimum of 2.
`y`	A numeric vector of y coordinates; minimum of 2.
`theta`	A numeric scalar with the angle of rotation in degrees; positive is clockwise.
`focus`	A numeric vector of x (first element) and y (second element) coordinates for the point around which `x` and `y` will rotate.

Details

Points are shifted to be centered at the focus, then rotated using a rotation matrix, then shifted back to original focus.

Value

A two-column data frame containing:

`x`	x coordinates
`y`	y coordinates

Note

This function is called from `crwInPolygon`

Author(s)

C. Holbrook (cholbrook@usgs.gov)

Examples

```
x <- runif(10,0,10)
y <- runif(10,0,10)
plot(x,y,type="b",pch=20)
foo <- rotatePoints(x, y, 20, c(5, 5))
points(foo$x,foo$y,type="b",pch=20,col="red")
```

`transmitAlongPath` *Simulate telemetry transmitter signals along a path*

Description

Simulate tag signal transmission along a pre-defined path (x, y coords) based on constant movement velocity, transmitter delay range, and duration of signal.

Usage

```
transmitAlongPath(path = NA, vel = 0.5, delayRng = c(60, 180),
  burstDur = 5)
```

Arguments

`path`	A two-column data frame with at least two rows and columns `x` and `y` with coordinates that define path.
`vel`	A numeric scalar with movement velocity along track; assumed constant.
`delayRng`	A 2-element numeric vector with minimum and maximum delay (time in seconds from end of one coded burst to beginning of next).
`burstDur`	A numeric scalar with duration (in seconds) of each coded burst (i.e., pulse train).

Details

Delays are drawn from uniform distribution defined by delay range. First, elapsed time in seconds at each node in path is calculated based on path length and velocity. Next, delays are simulated and burst durations are added toeach delay to determine the time of each signal transmission. Location of each signal transmission along the path is linearly interpolated.

Value

A two-column data frame containing:

`x`	x coordinates for start of each transmission
`y`	y coordinates for start of each transmission
`et`	elapsed time to start of each transmission

Note

This function was written to be called before `detectTransmissions`, which was designed to accept the result as input (`trnsLoc`).

Author(s)

C. Holbrook (cholbrook@usgs.gov)

Examples

```
mypath <- data.frame(x=seq(0,1000,100),y=seq(0,1000,100))
mytrns <- transmitAlongPath(mypath,vel=0.5,delayRng=c(60,180),burstDur=5.0)
plot(mypath,type="b")
points(mytrns,pch=20,col="red")
```

`vectorHeading` *Calculate direction (heading) of a vector (in degrees)*

Description

Calculate direction (heading) of a vector (in degrees)

Usage

```
vectorHeading(x, y)
```

Arguments

x	A numeric vector of x coordinates; minimum of 2.
y	A numeric vector of y coordinates; minimum of 2.

Details

Calculates direction (in degrees) for each of k-1 vectors, where k = length(x) - 1 = length(y). Lenghts of x and y must be equal.

Value

A numeric scalar with heading in degrees or a numeric vector of headings if `length(x) > 2`.

Note

This function is called from within `crwInPolygon`

Author(s)

C. Holbrook (cholbrook@usgs.gov)

Examples

```
x=c(2,4)
y=c(2,4)
vectorHeading(x, y)

x2=c(2,4,2)
y2=c(2,4,2)
vectorHeading(x2, y2)
```

vrl2csv *Convert Vemco VRL file(s) to CSV format (detection data only)*

Description

Convert detection data from a VEMCO VRL file(s) to comma-separated-values (CSV) format by invoking a system command in VUE (> 2.06; courtesy of Tim Stone, Vemco).

Usage

```
vrl2csv(vrl, outDir = NA, overwrite = TRUE, vueExePath = NA)
```

Arguments

vrl	A character string or vector with names of VRL file(s) or a single directory containing VRL files.
outDir	A character string directory where CSV files will be written. If NA (default) then file(s) will be written to the current working directory (e.g., getwd()).
overwrite	Logical. If TRUE (default), output CSV file(s) will overwrite existing CSV file(s) with same name in outDir. When FALSE, '_n' (i.e., _1, _2, etc.) will be appended to names of output files that already exist in outDir.
vueExePath	An optional character string with directory containing VUE.exe. If NA (default) then the path to VUE.exe must be added to the PATH environment variable of your system. See Note below.

Details

If vrl is a directory, then all VRL files in that directory will be converted to CSV. Otherwise, only those files specified in vrl will be converted. Each output CSV file will have same name as its source VRL file.

Value

A character vector with output directory and file name(s).

Note

Receiver event data are not exported because that functionality was not supported by the VUE system command at time of writing.

The path to VUE.exe must either be specified by vueExePath or added to the PATH environment variable of your system. To get the path to VUE.exe in Windows, right click on the icon, select "Properties", and then copy text in "Target" box.

To create a CSV for time-corrected VRL files, first time-correct each file using the VRL editor in VUE (under Tools menu). To speed up that process, uncheck the "Import" checkbox next to each filename, then run vrl2csv to create a CSV for each edited (e.g. time-corrected) VRL.

When using versions of VUE before 2.3, VUE can return an error code or warning message even if conversion was successful.

Author(s)

C. Holbrook (cholbrook@usgs.gov)

Examples

```
#get path to example VRL in this package
myVRL <- system.file("extdata", "VR2W_109924_20110718_1.vrl",
 package="glatos")
vrl2csv(dirname(myVRL)) #directory input
vrl2csv(myVRL) #file name input

#setting 'overwrite=FALSE' will make new file with '_n'added to name
vrl2csv(myVRL, overwrite=F)
```

Index