Package ‘yaps’

April 13, 2021

Title  Track Estimation using YAPS (Yet Another Positioning Solver)

Version  1.2.5

Description  Estimate tracks of animals tagged with acoustic transmitters. 'yaps' was introduced in 2017 as a transparent open-source tool to estimate positions of fish (and other aquatic animals) tagged with acoustic transmitters. Based on registrations of acoustic transmitters on hydrophones positioned in a fixed array, 'yaps' enables users to synchronize the collected data (i.e. correcting for drift in the internal clocks of the hydrophones/receivers) and subsequently to estimate tracks of the tagged animals. The paper introducing 'yaps' is available in open access at Baktoft, Gjelland, Økland & Thygesen (2017) <doi:10.1038/s41598-017-14278-z>. Also check out our cookbook with a completely worked through example at Baktoft, Gjelland, Økland, Rehage, Rodemann, Corujo, Viadero & Thygesen (2019) <DOI:10.1101/2019.12.16.877688>. Additional tutorials will eventually make their way onto the project website at <https://baktoft.github.io/yaps/>.

Depends  R (>= 3.5.0)

License  GPL-3

Encoding  UTF-8

LazyData  true

RoxygenNote  7.1.1

LinkingTo  Rcpp, TMB, RcppEigen

Imports  circular, cowplot, data.table, ggplot2, ggrepel, nloptr, plyr,
Rcpp, reshape2, splusTimeSeries, stats, tictoc, TMB, viridis,
zoo

Suggests  caTools, covr, knitr, rmarkdown, testthat (>= 2.1.0), vdiffr


BugReports  https://github.com/baktoft/yaps/issues

NeedsCompilation  yes

Author  Henrik Baktoft [cre, aut] (<https://orcid.org/0000-0002-3644-4960>),
Karl Gjelland [aut] (<https://orcid.org/0000-0003-4036-4207>),
Uffe H. Thygesen [aut] (<https://orcid.org/0000-0002-4311-6324>),
Finn Økland [aut] (<https://orcid.org/0000-0002-1938-5460>)

Maintainer  Henrik Baktoft <hba@aqua.dtu.dk>
Repository  CRAN
Date/Publication  2021-04-13 21:30:02 UTC

R topics documented:

alignBurstSeq .................................................. 2
applySync ........................................................ 3
checkInp .......................................................... 6
checkInpSync .................................................... 8
dat_align ........................................................ 10
fineTuneSyncModel ............................................. 11
getBbox ........................................................... 13
getInp ............................................................. 14
getInpSync ....................................................... 17
getSyncCoverage ................................................ 20
getSyncModel .................................................... 22
getToaYaps ....................................................... 25
plotBbox .......................................................... 27
plotSyncModelCheck ........................................... 28
plotSyncModelHydros .......................................... 29
plotSyncModelResids .......................................... 30
plotYaps .......................................................... 31
prepDetections ................................................... 32
runYaps ........................................................... 32
simHydros ........................................................ 35
simTelemetryTrack ............................................. 37
simToa ............................................................ 39
simTrueTrack .................................................... 41
ssu1 ............................................................... 43
tempToSs .......................................................... 44
testYaps ........................................................... 44

Index  46

_{alignBurstSeq}  
_{Align synced data with known burst sequence}

Description

Identifies where in the sequence of known burst intervals the detected data is from. Add extra columns to data.table containing ping index of the burst sequence (seq_ping_idx) and expected time of ping (seq_epo). Only to be used for 'random' burst interval data when you know the burst sequence.
applySync

Usage

alignBurstSeq(
    synced_dat,
    burst_seq,
    seq_lng_min = 10,
    rbi_min,
    rbi_max,
    plot_diag = TRUE
)

Arguments

synced_dat data.table obtained using applySync() on a detections_table
burst_seq Vector containing known burst sequence
seq_lng_min Minimum length of sequence of consecutive pings to use for the alignment. Finds first occurrence of sequence of this length in the data and compare to the known burst sequence
rbi_min, rbi_max Minimum and maximum burst interval of the transmitter. Used to identify sequence of consecutive pings in the data
plot_diag Logical indicating if visual diagnosis plots should be created.

Value
data.table like the input synced_dat, but with extra columns seq_ping_idx and seq_epo

Examples

# Align data from a tag with known random burst interval to the burst interval sequence # using the hald data included in 'yapsdata' (see ?yapsdata::hald for info).
synced_dat_1315 <- dat_align$synced_dat_1315
seq_1315 <- dat_align$seq_1315
rbi_min <- 60
rbi_max <- 120
aligned_dat <- alignBurstSeq(synced_dat=synced_dat_1315, burst_seq=seq_1315,
rbi_min=rbi_min, rbi_max=rbi_max, plot_diag=TRUE)

applySync

Apply sync model to toa matrix to obtain synced data

Description

Apply sync model to toa matrix to obtain synced data

Usage

applySync(toa, hydros = "", sync_model)
Arguments

- **toa**: Object containing data to be synchronized. Typically a `data.table` as e.g. `ssu1$detections`, but can also be a matrix `dim=(n_ping, n_hydo)`.
- **hydros**: `data.table` formatted as `ssu1$hydros`.
- **sync_model**: Synchronization model obtained using `getSyncModel()`.

Value

A `data.table` with the now synchronized time-of-arrivals in column `eposync`.

Examples

```r
library(yaps)
set.seed(42)

# # # Example using the ssu1 data included in package. See ?ssu1 for info.
# # # Set parameters to use in the sync model - these will differ per study
max_epo_diff <- 120
min_hydros <- 2
time_keeper_idx <- 5
fixed_hydros_idx <- c(2:3, 6, 8, 11, 13:17)
n_offset_day <- 2
n_ss_day <- 2
keep_rate <- 20

# # # Get input data ready for getSyncModel()
inp_sync <- getInpSync(sync_dat=ssu1, max_epo_diff, min_hydros, time_keeper_idx,
                        fixed_hydros_idx, n_offset_day, n_ss_day, keep_rate=keep_rate, silent_check=TRUE)

# # # Check that inp_sync is ok
checkInpSync(inp_sync, silent_check=FALSE)

# # # Also take a look at coverage of the sync data
getSyncCoverage(inp_sync, plot=TRUE)

# # # Fit the sync model
sync_model <- getSyncModel(inp_sync, silent=TRUE, max_iter=200, tmb_smartsearch = TRUE)

# # # On some systems it might work better, if we disable the smartsearch feature in TMB
# # # To do so, set tmb_smartsearch = TRUE in getSyncModel()

# # # Visualize the resulting sync model
plotSyncModelResids(sync_model, by = "overall")
plotSyncModelResids(sync_model, by = "quantiles")
plotSyncModelResids(sync_model, by = "sync_tag")
plotSyncModelResids(sync_model, by = "hydro")
plotSyncModelResids(sync_model, by = "temporal_hydro")
plotSyncModelResids(sync_model, by = "temporal_sync_tag")

# # # If the above plots show outliers, sync_model can be fine tuned by excluding these.
```
applySync

# # # Use fineTuneSyncModel() for this.
# # # This should typically be done sequentially using eps_thresholds of e.g. 1E4, 1E3, 1E2, 1E2
sync_model <- fineTuneSyncModel(sync_model, eps_threshold=1E3, silent=TRUE)
sync_model <- fineTuneSyncModel(sync_model, eps_threshold=1E2, silent=TRUE)

# # # Apply the sync_model to detections data.
detections_synced <- applySync(toa=ssu1$detections, hydros=ssu1$hydros, sync_model)

# # # Prepare data for running yaps
hydros_yaps <- data.table::data.table(sync_model$p$TRUE_H)
colnames(hydros_yaps) <- c('hx', 'hy', 'hz')
focal_tag <- 15266
rbi_min <- 20
rbi_max <- 40
synced_dat <- detections_synced[tag == focal_tag]
toa <- getToaYaps(synced_dat=synced_dat, hydros=hydros_yaps, pingType='rbi',
rbi_min=rbi_min, rbi_max=rbi_max)
bbox <- getBbox(hydros_yaps, buffer=50, pen=1e6)
inp <- getInp(hydros_yaps, toa, E_dist="Mixture", n_ss=5, pingType="rbi",
   sdInits=1, rbi_min=rbi_min, rbi_max=rbi_max, ss_data_what="est", ss_data=0, bbox=bbox)

# # # Check that inp is ok
checkInp(inp)

# # # Run yaps on the prepared data to estimate track
yaps_out <- runYaps(inp, silent=TRUE, tmb_smartsearch=TRUE, maxIter=5000)

# # # Plot the results and compare to "the truth" obtained using gps
oldpar <- par(no.readonly = TRUE)
par(mfrow=c(2,2))
plot(hy~hx, data=hydros_yaps, asp=1, xlab="UTM X", ylab="UTM Y", pch=20, col="green")
lines(utm_y~utm_x, data=ssu1$gps, col="blue", lwd=2)
lines(y~x, data=yaps_out$track, col="red")

plot(utm_x~ts, data=ssu1$gps, col="blue", type="l", lwd=2)
points(x~top, data=yaps_out$track, col="red")
lines(x~top, data=yaps_out$track, col="red")
lines(x-2*x_sd~top, data=yaps_out$track, col="red", lty=2)
lines(x+2*x_sd~top, data=yaps_out$track, col="red", lty=2)

plot(utm_y~ts, data=ssu1$gps, col="blue", type="l", lwd=2)
points(y~top, data=yaps_out$track, col="red")
lines(y~top, data=yaps_out$track, col="red")
lines(y-2*y_sd~top, data=yaps_out$track, col="red", lty=2)
lines(y+2*y_sd~top, data=yaps_out$track, col="red", lty=2)

plot(nobs~top, data=yaps_out$track, type="p", main="#detecting hydros per ping")
lines(caTools::runmean(nobs, k=10)-top, data=yaps_out$track, col="orange", lwd=2)
par(oldpar)
checkInp

Check consistency of inp object obtained from getInp()

Description

Check consistency of inp object obtained from getInp()

Usage

checkInp(inp)

Arguments

inp Object obtained using getInp()

Value

No return value, but prints errors/warnings if issues with inp is detected.

Examples

library(yaps)
set.seed(42)

### Example using the ssu1 data included in package. See ?ssu1 for info.
### Set parameters to use in the sync model - these will differ per study
max_epo_diff <- 120
min_hydros <- 2
time_keeper_idx <- 5
fixed_hydros_idx <- c(2:3, 6, 8, 11, 13:17)
offset_day <- 2
n_ss_day <- 2
keep_rate <- 20

### Get input data ready for getSyncModel()
inp_sync <- getInpSync(sync_dat=ssu1, max_epo_diff, min_hydros, time_keeper_idx, fixed_hydros_idx, n_offset_day, n_ss_day, keep_rate=keep_rate, silent_check=TRUE)

### Check that inp_sync is ok
checkInpSync(inp_sync, silent_check=FALSE)

### Also take a look at coverage of the sync data
getSyncCoverage(inp_sync, plot=TRUE)

### Fit the sync model
sync_model <- getSyncModel(inp_sync, silent=TRUE, max_iter=200, tmb_smartsearch = TRUE)

### On some systems it might work better, if we disbale the smartsearch feature in TMB
### To do so, set tmb_smartsearch = FALSE in getSyncModel()
# Visualize the resulting sync model
plotSyncModelResids(sync_model, by = "overall")
plotSyncModelResids(sync_model, by = "quantiles")
plotSyncModelResids(sync_model, by = "sync_tag")
plotSyncModelResids(sync_model, by = "hydro")
plotSyncModelResids(sync_model, by = "temporal_hydro")
plotSyncModelResids(sync_model, by = "temporal_sync_tag")

# If the above plots show outliers, sync_model can be fine tuned by excluding these.
# Use fineTuneSyncModel() for this.
# This should typically be done sequentially using eps_thresholds of e.g. 1E4, 1E3, 1E2, 1E2
sync_model <- fineTuneSyncModel(sync_model, eps_threshold=1E3, silent=TRUE)
sync_model <- fineTuneSyncModel(sync_model, eps_threshold=1E2, silent=TRUE)

# Apply the sync_model to detections data.
detections_synced <- applySync(toa=ssu1$detections, hydros=ssu1$hydros, sync_model)

# Prepare data for running yaps
hydros_yaps <- data.table::data.table(sync_model$pl$TRUE_H)
colnames(hydros_yaps) <- c('hx','hy','hz')
focal_tag <- 15266
rbi_min <- 20
rbi_max <- 40
synced_dat <- detections_synced[tag == focal_tag]
toa <- getToaYaps(synced_dat=synced_dat, hydros=hydros_yaps, pingType='rbi',
rbi_min=rbi_min, rbi_max=rbi_max)
bbox <- getBbox(hydros_yaps, buffer=50, pen=1e6)
inp <- getInp(hydros_yaps, toa, E_dist="Mixture", n_ss=5, pingType="rbi",
sdInits=1, rbi_min=rbi_min, rbi_max=rbi_max, ss_data_what="est", ss_data=0, bbox=bbox)

# Check that inp is ok
checkInp(inp)

# Run yaps on the prepared data to estimate track
yaps_out <- runYaps(inp, silent=TRUE, tmb_smartsearch=TRUE, maxIter=5000)

# Plot the results and compare to "the truth" obtained using gps
oldpar <- par(no.readonly = TRUE)
par(mfrow=c(2,2))
plot(hy~hx, data=hydros_yaps, asp=1, xlab="UTM X", ylab="UTM Y", pch=20, col="green")
lines(utm_y~utm_x, data=ssu1$gps, col="blue", lwd=2)
lines(y~x, data=yaps_out$track, col="red")

plot(utm_x~ts, data=ssu1$gps, col="blue", type="l", lwd=2)
points(x~top, data=yaps_out$track, col="blue", lwd=2)
lines(x~top, data=yaps_out$track, col="red")
lines(x~2*x_sd~top, data=yaps_out$track, col="red", lty=2)
lines(x~2*x_sd~top, data=yaps_out$track, col="red", lty=2)

plot(utm_y~ts, data=ssu1$gps, col="blue", type="l", lwd=2)
points(y~top, data=yaps_out$track, col="red")

oldpar
lines(y~top, data=yaps_out$track, col="red")
lines(y-2*y_sd~top, data=yaps_out$track, col="red", lty=2)
lines(y+2*y_sd~top, data=yaps_out$track, col="red", lty=2)

plot(nobs~top, data=yaps_out$track, type="p", main="#detecting hydros per ping")
lines(caTools::runmean(nobs, k=10)~top, data=yaps_out$track, col="orange", lwd=2)
par(oldpar)

---

checkInpSync

Check consistency of inp_sync object obtained from getInpSync()

Description

Check consistency of inp_sync object obtained from getInpSync()

Usage

checkInpSync(inp_sync, silent_check)

Arguments

inp_sync Object obtained using getInpSync()
silent_check Logical whether to get output from checkInpSync(). Default is FALSE

Value

No return value, but prints errors/warnings if issues with inp_sync is detected.

Examples

library(yaps)
set.seed(42)

### Example using the ssu1 data included in package. See ?ssu1 for info.
### Set parameters to use in the sync model - these will differ per study
max_epo_diff <- 120
min_hydros <- 2
time_keeper_idx <- 5
fixed_hydros_idx <- c(2:3, 6, 8, 11, 13:17)
n_offset_day <- 2
n_ss_day <- 2
keep_rate <- 20

### Get input data ready for getSyncModel()
inp_sync <- getInpSync(sync_dat=ssu1, max_epo_diff, min_hydros, time_keeper_idx,
fixed_hydros_idx, n_offset_day, n_ss_day, keep_rate=keep_rate, silent_check=TRUE)
checkInpSync

```r
## # # Check that inp_sync is ok
checkInpSync(inp_sync, silent_check=FALSE)

## # # Also take a look at coverage of the sync data
getSyncCoverage(inp_sync, plot=TRUE)

## # # Fit the sync model
sync_model <- getSyncModel(inp_sync, silent=TRUE, max_iter=200, tmb_smartsearch = TRUE)

## # # On some systems it might work better, if we disable the smartsearch feature in TMB
## # # To do so, set tmb_smartsearch = FALSE in getSyncModel()

## # # Visualize the resulting sync model
plotSyncModelResids(sync_model, by = "overall")
plotSyncModelResids(sync_model, by = "quantiles")
plotSyncModelResids(sync_model, by = "sync_tag")
plotSyncModelResids(sync_model, by = "hydro")
plotSyncModelResids(sync_model, by = "temporal_hydro")
plotSyncModelResids(sync_model, by = "temporal_sync_tag")

## # # If the above plots show outliers, sync_model can be fine tuned by excluding these.
## # # Use fineTuneSyncModel() for this.
## # # This should typically be done sequentially using eps_thresholds of e.g. 1E4, 1E3, 1E2, 1E2
sync_model <- fineTuneSyncModel(sync_model, eps_threshold=1E3, silent=TRUE)
sync_model <- fineTuneSyncModel(sync_model, eps_threshold=1E2, silent=TRUE)

## # # Apply the sync_model to detections data.
detections_synced <- applySync(toa=ssu1$detections, hydros=ssu1$hydros, sync_model)

## # # Prepare data for running yaps
hydros_yaps <- data.table::data.table(sync_model$pl$TRUE_H)
colnames(hydros_yaps) <- c("hx", "hy", "hz")
focal_tag <- 15266
rbi_min <- 20
rbi_max <- 40
detections_synced <- detections_synced[tag == focal_tag]
toa <- getToaYaps(detections_synced, hydros=hydros_yaps, pingType="rbi",
rbi_min=rbi_min, rbi_max=rbi_max)
bbox <- getBbox(hydros_yaps, buffer=50, pen=1e6)
inp <- getInp(hydros_yaps, toa, E_dist="Mixture", n_ss=5, pingType="rbi",
sdInits=1, rbi_min=rbi_min, rbi_max=rbi_max, ss_data_what="est", ss_data=0, bbox=bbox)

## # # Check that inp is ok
checkInp(inp)

## # # Run yaps on the prepared data to estimate track
yaps_out <- runYaps(inp, silent=TRUE, tmb_smartsearch=TRUE, maxIter=5000)

## # # Plot the results and compare to "the truth" obtained using gps
oldpar <- par(no.readonly = TRUE)
par(mfrow=c(2,2))
plot(hy~hx, data=hydros_yaps, asp=1, xlab="UTM X", ylab="UTM Y", pch=20, col="green")```
Example data for showcasing yaps function `alignBurstSeq()`

**Description**

Function `alignBurstSeq()` is used to align synced detection data with a sequence of known random burst intervals (BI).

This step is needed to take advantage of the extra information available when working with random BI data with a known sequence.

This small sample is obtained from the accompanying data package yapsdata.

**Usage**

dat_align

**Format**

A list containing 2 items:

- `synced_dat_1315` data.table containing synced detections of tag 1315.
- `synced_dat_1315` vector of small part of the complete sequence of known random BIs.
fineTuneSyncModel

Fine-tune an already fitted sync_model Wrapper function to re-run getSyncModel() using the same data, but excluding outliers. Note dimensions of data might change if eps_threshold results in empty rows in the TOA-matrix.

Description

Fine-tune an already fitted sync_model Wrapper function to re-run getSyncModel() using the same data, but excluding outliers. Note dimensions of data might change if eps_threshold results in empty rows in the TOA-matrix.

Usage

\[
\text{fineTuneSyncModel(sync_model, eps_threshold, silent = TRUE)}
\]

Arguments

- \text{sync_model} \quad \text{sync_model obtained using getSyncModel()}
- \text{eps_threshold} \quad \text{Maximum value of residual measured in meter assuming speed of sound = 1450 m/s}
- \text{silent} \quad \text{logical whether to make getSyncModel() silent}

Value

Fine tuned sync_model. See ?getSyncModel for more info.

Examples

library(yaps)
set.seed(42)

# # # Example using the ssu1 data included in package. See ?ssu1 for info.
# # # Set parameters to use in the sync model - these will differ per study
max_epo_diff <- 120
min_hydros <- 2
time_keeper_idx <- 5
fixed_hydros_idx <- c(2:3, 6, 8, 11, 13:17)
n_offset_day <- 2
n_ss_day <- 2
keep_rate <- 20

# # # Get input data ready for getSyncModel()
inp_sync <- getInpSync(sync_dat=ssu1, max_epo_diff, min_hydros, time_keeper_idx,
fixed_hydros_idx, n_offset_day, n_ss_day, keep_rate=keep_rate, silent_check=TRUE)

# # # Check that inp_sync is ok
checkInpSync(inp_sync, silent_check=FALSE)

# # # Also take a look at coverage of the sync data
getSyncCoverage(inp_sync, plot=TRUE)

# # # Fit the sync model
sync_model <- getSyncModel(inp_sync, silent=TRUE, max_iter=200, tmb_smartsearch = TRUE)

# # # On some systems it might work better, if we disable the smartsearch feature in TMB
# # # To do so, set tmb_smartsearch = FALSE in getSyncModel()

# # # Visualize the resulting sync model
plotSyncModelResids(sync_model, by = "overall")
plotSyncModelResids(sync_model, by = "quantiles")
plotSyncModelResids(sync_model, by = "sync_tag")
plotSyncModelResids(sync_model, by = "hydro")
plotSyncModelResids(sync_model, by = "temporal_hydro")
plotSyncModelResids(sync_model, by = "temporal_sync_tag")

# # # If the above plots show outliers, sync_model can be fine tuned by excluding these.
# # # Use fineTuneSyncModel() for this.
# # # This should typically be done sequentially using eps_thresholds of e.g. 1E4, 1E3, 1E2, 1E2
sync_model <- fineTuneSyncModel(sync_model, eps_threshold=1E3, silent=TRUE)
sync_model <- fineTuneSyncModel(sync_model, eps_threshold=1E2, silent=TRUE)

# # # Apply the sync_model to detections data.
detections_synced <- applySync(toa=ssu1$detections, hydros=ssu1$hydros, sync_model)

# # # Prepare data for running yaps
hydros_yaps <- data.table::data.table(sync_model$pl$TRUE_H)
colnames(hydros_yaps) <- c('hx', 'hy', 'hz')
focal_tag <- 15266
rbi_min <- 20
rbi_max <- 40
synced_dat <- detections_synced[tag == focal_tag]
toa <- getToaYaps(synced_dat=synced_dat, hydros=hydros_yaps, pingType='rbi',
    rbi_min=rbi_min, rbi_max=rbi_max)
bbox <- getBbox(hydros_yaps, buffer=50, pen=1e6)
inp <- getInp(hydros_yaps, toa, E_dist="Mixture", n_ss=5, pingType="rbi",
    sdInits=1, rbi_min=rbi_min, rbi_max=rbi_max, ss_data_what="est", ss_data=0, bbox=bbox)

# # # Check that inp is ok
checkInp(inp)

# # # Run yaps on the prepared data to estimate track
yaps_out <- runYaps(inp, silent=TRUE, tmb_smartsearch=TRUE, maxIter=5000)

# # # Plot the results and compare to "the truth" obtained using gps
oldpar <- par(no.readonly = TRUE)
par(mfrow=c(2,2))
plot(hy~hx, data=hydros_yaps, asp=1, xlab="UTM X", ylab="UTM Y", pch=20, col="green")
lines(utm_y~utm_x, data=ssu1$gps, col="blue", lwd=2)
getBbox

Get a standard bounding box to impose spatial constraints

Description

Standard is a rectangle based on coordinates of outer hydros + the buffer in meters

Usage

getBbox(hydros, buffer = 100, eps = 0.001, pen = 1e+06)

Arguments

hydros Dataframe from simHydros() or Dataframe with columns hx and hy containing positions of the receivers. Translate the coordinates to get the grid centre close to (0,0).
buffer Number of meters the spatial domain extends beyond the outer hydros.
eps Specifies how well-defined the borders are (eps=1E-2 is very sharp, eps=100 is very soft).
pen Specifies the penalty multiplier.

Value

Vector of length 6: c(x_min, x_max, y_min, y_max, eps, pen). Limits are given in UTM coordinates.
getInp <- getInp(hydros, toa, E_dist, n_ss, pingType, sdInits = 1, rbi_min = 0, rbi_max = 0, ss_data_what = "est", ss_data = 0, biTable = NULL, z_vec = NULL, bbox = NULL)

Arguments

- **hydros**: Dataframe from simHydros() or Dataframe with columns hx and hy containing positions of the receivers. Translate the coordinates to get the grid centre close to (0;0).
- **toa**: TOA-matrix: matrix with receivers in rows and detections in columns. Make sure that the receivers are in the same order as in hydros, and that the matrix is very regular: one ping per column (include empty columns if a ping is not detected).
- **E_dist**: Which distribution to use in the model - "Gaus" = Gaussian, "Mixture" = mixture of Gaussian and t or "t" = pure t-distribution
- **n_ss**: Number of soundspeed estimates: one estimate per hour is usually enough
- **pingType**: Type of transmitter to simulate - either stable burst interval ("sbi"), random burst interval ("rbi") or random burst interval but where the random sequence is known a priori

Description

Wrapper-function to compile a list of input needed to run TMB

Usage

getInp(hydros, toa, E_dist, n_ss, pingType, sdInits = 1, rbi_min = 0, rbi_max = 0, ss_data_what = "est", ss_data = 0, biTable = NULL, z_vec = NULL, bbox = NULL)
**getInp**

**sdInits**
If >0 initial values will be randomized around the normally fixed value using rnorm(length(inits), mean=inits, sd=sdInits)

**rbi_min, rbi_max**
Minimum and maximum BI for random burst interval transmitters

**ss_data_what**
What speed of sound (ss) data to be used. Default ss_data_what='est': ss is estimated by the model. Alternatively, if ss_data_what='data': ss_data must be provided and length(ss_data) == ncol(toa)

**ss_data**
Vector of ss-data to be used if ss_data_what = 'est'. Otherwise ss_data <- 0 (default)

**biTable**
Table of known burst intervals. Only used when pingType == "pbi". Default=NULL

**z_vec**
Vector of known depth values (positive real). Default=NULL is which case no 3D is assumed. If z_vec = "est" depth will be estimated.

**bbox**
Spatial constraints in the form of a bounding box. See ?getBbox for details.

**Value**
List of input data ready for use in runYaps()

**Examples**

library(yaps)
set.seed(42)

```r
# # # Example using the ssu1 data included in package. See ?ssu1 for info.
# # # Set parameters to use in the sync model - these will differ per study
max_epo_diff <- 120
min_hydros <- 2
time_keeper_idx <- 5
fixed_hydros_idx <- c(2:3, 6, 8, 11, 13:17)
n_offset_day <- 2
n_ss_day <- 2
keep_rate <- 20

# # # Get input data ready for getSyncModel()
inp_sync <- getInpSync(sync_dat=ssu1, max_epo_diff, min_hydros, time_keeper_idx,
fixed_hydros_idx, n_offset_day, n_ss_day, keep_rate, silent_check=TRUE)

# # # Check that inp_sync is ok
checkInpSync(inp_sync, silent_check=FALSE)

# # # Also take a look at coverage of the sync data
getSyncCoverage(inp_sync, plot=TRUE)

# # # Fit the sync model
sync_model <- getSyncModel(inp_sync, silent=TRUE, max_iter=200, tmb_smartsearch = TRUE)

# # # On some systems it might work better, if we disable the smartsearch feature in TMB
# # # To do so, set tmb_smartsearch = FALSE in getSyncModel()
```
### Visualize the resulting sync model

```r
plotSyncModelResids(sync_model, by = "overall")
plotSyncModelResids(sync_model, by = "quantiles")
plotSyncModelResids(sync_model, by = "sync_tag")
plotSyncModelResids(sync_model, by = "hydro")
plotSyncModelResids(sync_model, by = "temporal_hydro")
plotSyncModelResids(sync_model, by = "temporal_sync_tag")
```

### If the above plots show outliers, sync_model can be fine tuned by excluding these.

Use `fineTuneSyncModel()` for this.

This should typically be done sequentially using eps_thresholds of e.g. 1E4, 1E3, 1E2, 1E2

```r
sync_model <- fineTuneSyncModel(sync_model, eps_threshold=1E3, silent=TRUE)
sync_model <- fineTuneSyncModel(sync_model, eps_threshold=1E2, silent=TRUE)
```

### Apply the sync_model to detections data.

```r
detections_synced <- applySync(toa=ssu1$detections, hydros=ssu1$hydros, sync_model)
```

### Prepare data for running yaps

```r
hydros_yaps <- data.table::data.table(sync_model$pl$TRUE_H)
colnames(hydros_yaps) <- c('hx', 'hy', 'hz')
focal_tag <- 15266
rbi_min <- 20
rbi_max <- 40
synced_dat <- detections_synced[tag == focal_tag]
toa <- getToaYaps(synced_dat=synced_dat, hydros=hydros_yaps, pingType='rbi',
rbi_min=rbi_min, rbi_max=rbi_max)
bbox <- getBbox(hydros_yaps, buffer=50, pen=1e6)
inp <- getInp(hydros_yaps, toa, E_dist="Mixture", n_ss=5, pingType="rbi",
    sdInits=1, rbi_min=rbi_min, rbi_max=rbi_max, ss_data_what="est", ss_data=0, bbox=bbox)
```

### Check that inp is ok

```r
checkInp(inp)
```

### Run yaps on the prepared data to estimate track

```r
yaps_out <- runYaps(inp, silent=TRUE, tmb_smartsearch=TRUE, maxIter=5000)
```

### Plot the results and compare to "the truth" obtained using gps

```r
oldpar <- par(no.readonly = TRUE)
par(mfrow=c(2,2))
plot(hy~hx, data=hydros_yaps, asp=1, xlab="UTM X", ylab="UTM Y", pch=20, col="green")
lines(utm_y~utm_x, data=ssu1$gps, col="blue", lwd=2)
lines(y~x, data=yaps_out$track, col="red")

plot(utm_x~ts, data=ssu1$gps, col="blue", type="l", lwd=2)
points(x~top, data=yaps_out$track, col="1", lwd=2)
lines(x~top, data=yaps_out$track, col="red")
lines(x~2*x.sd~top, data=yaps_out$track, col="red", lty=2)
lines(x~/2*x.sd~top, data=yaps_out$track, col="red", lty=2)
plot(utm_y~ts, data=ssu1$gps, col="blue", type="l", lwd=2)
points(y~top, data=yaps_out$track, col="red")
```
getInpSync

Get object inp for synchronization

Description
Get object inp for synchronization

Usage
getInpSync(
  sync_dat,
  max_epo_diff,
  min_hydros,
  time_keeper_idx,
  fixed_hydros_idx,
  n_offset_day,
  n_ss_day,
  keep_rate = 1,
  excl_self_detect = TRUE,
  lin_corr_coeffs = NA,
  ss_data_what = "est",
  ss_data = c(0),
  silent_check = FALSE
)

Arguments
sync_dat List containing data.tables with hydrophone information and detections. See e.g. ?ssu1 for example
max_epo_diff Sets the upper threshold for differences in TOA of sync tags. Best parameter value depends on burst rate of sync tags and how far apart the internal clocks of the hydros are prior to synchronization. A bit less than half of minimum sync tag burst rate is a good starting choice.
min_hydros Sets the lower threshold of how many hydrophones need to detect each sync tag ping in order to be included in the sync process. Should be as high as possible while observing that all hydrophones are contributing. If too low, isolated hydrophones risk falling out completely. Future versions will work towards automising this.
getTimeKeeperIndex

Index of the hydrophone to use as time keeper. Could e.g. be the one with smallest overall clock-drift.

getFixedHdyrosIdx

Vector of hydro id’s for all hydrophones where the position is assumed to be known with adequate accuracy and precision. Include as many as possible as fixed hydros to reduce overall computation time and reduce overall variability. As a bare minimum two hydros need to be fixed, but we strongly advice to use more than two.

getNOffsetDay

Specifies the number of hydrophone specific quadratic polynomials to use per day. For PPM based systems, 1 or 2 is often adequate.

getNSSDay

Specifies number of speed of sound to estimate per day if no ss data is supplied. It is recommended to use logged water temperature instead. However, estimating SS gives an extra option for sanity-checking the final sync-model.

getKeepRate

Syncing large data sets can take a really long time. However, there is typically an excess number of sync tag detections and a sub-sample is typically enough for good synchronization. This parameter EITHER specifies a proportion (0-1) of data to keep when sub-sampling OR (if keep_rate > 10) number of pings (approximate) to keep in each hydro X offset_idx combination if enough exists.

getExclSelfDetect

Logical whether to excluded detections of sync tags on the hydros they are collocated with. Sometimes self detections can introduce excessive residuals in the sync model in which case they should be excluded.

getLinCorrCoeffs

Matrix of coefficients used for pre-sync linear correction. dim(lin_corr_coeffs)=(#hydros, 2).

getSSDataWhat

Indicates whether to estimate ("est") speed of sound or to use data based on logged water temperature ("data").

getSSData

data.table containing timestamp and speed of sound for the entire period to by synchronised. Must contain columns 'ts' (POSIXct timestamp) and 'ss' speed of sound in m/s (typical values range 1400 - 1550).

getSilentCheck

Logical whether to get output from checkInpSync(). Default is FALSE

Value

List of input data ready for use in getSyncModel()

Examples

```r
library(yaps)
set.seed(42)

# # Example using the ssu1 data included in package. See ?ssu1 for info.
# # Set parameters to use in the sync model - these will differ per study
max_epo_diff <- 120
min_hydros <- 2
time_keeper_idx <- 5
fixed_hydros_idx <- c(2:3, 6, 8, 11, 13:17)
```
n_offset_day <- 2
n_ss_day <- 2
keep_rate <- 20

# # # Get input data ready for getSyncModel()
inp_sync <- getInpSync(sync_dat=ssu1, max_epo_diff, min_hydros, time_keeper_idx, fixed_hydros_idx, n_offset_day, n_ss_day, keep_rate=keep_rate, silent_check=TRUE)

# # # Check that inp_sync is ok
checkInpSync(inp_sync, silent_check=FALSE)

# # # Also take a look at coverage of the sync data
getSyncCoverage(inp_sync, plot=TRUE)

# # # Fit the sync model
sync_model <- getSyncModel(inp_sync, silent=TRUE, max_iter=200, tmb_smartsearch = TRUE)

# # # On some systems it might work better, if we disable the smartsearch feature in TMB
# # # To do so, set tmb_smartsearch = FALSE in getSyncModel()

# # # Visualize the resulting sync model
plotSyncModelResids(sync_model, by = "overall")
plotSyncModelResids(sync_model, by = "quantiles")
plotSyncModelResids(sync_model, by = "sync_tag")
plotSyncModelResids(sync_model, by = "hydro")
plotSyncModelResids(sync_model, by = "temporal_hydro")
plotSyncModelResids(sync_model, by = "temporal_sync_tag")

# # # If the above plots show outliers, sync_model can be fine tuned by excluding these.
# # # Use fineTuneSyncModel() for this.
# # # This should typically be done sequentially using eps_thresholds of e.g. 1E4, 1E3, 1E2
sync_model <- fineTuneSyncModel(sync_model, eps_threshold=1E3, silent=TRUE)
sync_model <- fineTuneSyncModel(sync_model, eps_threshold=1E2, silent=TRUE)

# # # Apply the sync_model to detections data.
detections_synced <- applySync(toa=ssu1$detections, hydros=ssu1$hydros, sync_model)

# # # Prepare data for running yaps
hydros_yaps <- data.table::data.table(sync_model$pl$TRUE_H)
colnames(hydros_yaps) <- c('hx', 'hy', 'hz')
focal_tag <- 15266
rbi_min <- 20
rbi_max <- 40
synced_dat <- detections_synced[tag == focal_tag]
toa <- getToaYaps(synced_dat, hydros=hydros_yaps, pingType='rbi', rbi_min=rbi_min, rbi_max=rbi_max)
bbox <- getBbox(hydros_yaps, buffer=50, pen=1e6)
inp <- getInp(hydros_yaps, toa, E_dist="Mixture", n_ss=5, pingType="rbi", sdInits=1, rbi_min=rbi_min, rbi_max=rbi_max, ss_data_what="est", ss_data=0, bbox=bbox)

# # # Check that inp is ok
checkInp(inp)
## Run yaps on the prepared data to estimate track

```r
yaps_out <- runYaps(inp, silent=TRUE, tmb_smartsearch=TRUE, maxIter=5000)
```

## Plot the results and compare to "the truth" obtained using gps

```r
oldpar <- par(no.readonly = TRUE)
par(mfrow=c(2,2))
plot(hy~hx, data=hydrs_yaps, asp=1, xlab="UTM X", ylab="UTM Y", pch=20, col="green")
lines(utm_y~utm_x, data=ssu1$gps, col="blue", lwd=2)
lines(y~x, data=yaps_out$track, col="red")

plot(utm_x~ts, data=ssu1$gps, col="blue", type="l", lwd=2)
points(x~top, data=yaps_out$track, col="red")
lines(x~2*x_sd~top, data=yaps_out$track, col="red", lty=2)
lines(x+2*x_sd~top, data=yaps_out$track, col="red", lty=2)

plot(utm_y~ts, data=ssu1$gps, col="blue", type="l", lwd=2)
points(y~top, data=yaps_out$track, col="red")
lines(y~2*y_sd~top, data=yaps_out$track, col="red", lty=2)
lines(y+2*y_sd~top, data=yaps_out$track, col="red", lty=2)

plot(nobs~top, data=yaps_out$track, type="p", main="#detecting hydros per ping")
lines(caTools::runmean(nobs, k=10)-top, data=yaps_out$track, col="orange", lwd=2)
par(oldpar)
```

---

**getSyncCoverage**

Quick overview to check if all hydros have enough data within each offset period.

### Description

Quick overview to check if all hydros have enough data within each offset period.

### Usage

```r
getSyncCoverage(inp_sync, plot = FALSE)
```

### Arguments

- **inp_sync**: Object obtained using `getInpSync()`
- **plot**: Logical indicating whether to plot a visual or not.

### Value

A data.table containing number of pings included in each hydro x offset combination.
Examples

library(yaps)
set.seed(42)

## Example using the ssu1 data included in package. See ?ssu1 for info.
## Set parameters to use in the sync model - these will differ per study
max_epo_diff <- 120
min_hydros <- 2
time_keeper_idx <- 5
fixed_hydros_idx <- c(2:3, 6, 8, 11, 13:17)
n_offset_day <- 2
n_ss_day <- 2
keep_rate <- 20

## Get input data ready for getSyncModel()
inp_sync <- getInpSync(sync_dat=ssu1, max_epo_diff, min_hydros, time_keeper_idx,
                      fixed_hydros_idx, n_offset_day, n_ss_day, keep_rate=keep_rate, silent_check=TRUE)

## Check that inp_sync is ok
checkInpSync(inp_sync, silent_check=FALSE)

## Also take a look at coverage of the sync data
getSyncCoverage(inp_sync, plot=TRUE)

## Fit the sync model
sync_model <- getSyncModel(inp_sync, silent=TRUE, max_iter=200, tmb_smartsearch = TRUE)

## On some systems it might work better, if we disable the smartsearch feature in TMB
## To do so, set tmb_smartsearch = FALSE in getSyncModel()

## Visualize the resulting sync model
plotSyncModelResids(sync_model, by = "overall")
plotSyncModelResids(sync_model, by = "quantiles")
plotSyncModelResids(sync_model, by = "sync_tag")
plotSyncModelResids(sync_model, by = "hydro")
plotSyncModelResids(sync_model, by = "temporal_hydro")
plotSyncModelResids(sync_model, by = "temporal_sync_tag")

## If the above plots show outliers, sync_model can be fine tuned by excluding these.
## Use fineTuneSyncModel() for this.
## This should typically be done sequentially using eps_thresholds of e.g. 1E4, 1E3, 1E2
sync_model <- fineTuneSyncModel(sync_model, eps_threshold=1E3, silent=TRUE)
sync_model <- fineTuneSyncModel(sync_model, eps_threshold=1E2, silent=TRUE)

## Apply the sync_model to detections data.
detections_synced <- applySync(toa=ssu1$detections, hydros=ssu1$hydros, sync_model)

## Prepare data for running yaps
hydros_yaps <- data.table::data.table(sync_model$pl$TRUE_H)
colnames(hydros_yaps) <- c('hx','hy','hz')
focal_tag <- 15266
rbi_min <- 20
rbi_max <- 40
synced_dat <- detections_synced[tag == focal_tag]
toa <- getToaYaps(synced_dat=synced_dat, hydros=hydros_yaps, pingType='rbi',
               rbi_min=rbi_min, rbi_max=rbi_max)
bbox <- getBbox(hydros_yaps, buffer=50, pen=1e6)
inp <- getInp(hydros_yaps, toa, E_dist="Mixture", n_ss=5, pingType="rbi",
               sdInitial=1, rbi_min=rbi_min, rbi_max=rbi_max, ss_data_what="est", ss_data=0, bbox=bbox)

## # Check that inp is ok
checkInp(inp)

## # Run yaps on the prepared data to estimate track
yaps_out <- runYaps(inp, silent=TRUE, tmb_smartsearch=TRUE, maxIter=5000)

## # Plot the results and compare to “the truth” obtained using gps
oldpar <- par(no.readonly = TRUE)
par(mfrow=c(2,2))
plot(hy~hx, data=hydros_yaps, asp=1, xlab="UTM X", ylab="UTM Y", pch=20, col="green")
lines(utm_y~utm_x, data=ssu1$gps, col="blue", lwd=2)
lines(y~x, data=yaps_out$track, col="red")

plot(utm_x~ts, data=ssu1$gps, col="blue", type="l", lwd=2)
points(x~top, data=yaps_out$track, col="red")
lines(x~top, data=yaps_out$track, col="red")
lines(x+2*x_sd~top, data=yaps_out$track, col="red", lty=2)
lines(x-2*x_sd~top, data=yaps_out$track, col="red", lty=2)

plot(utm_y~ts, data=ssu1$gps, col="blue", type="l", lwd=2)
points(y~top, data=yaps_out$track, col="red")
lines(y~top, data=yaps_out$track, col="red")
lines(y+2*y_sd~top, data=yaps_out$track, col="red", lty=2)
lines(y-2*y_sd~top, data=yaps_out$track, col="red", lty=2)

plot(nobs~top, data=yaps_out$track, type="p", main="#detecting hydros per ping")
lines(caTools::runmean(nobs, k=10)+top, data=yaps_out$track, col="orange", lwd=2)
par(oldpar)

getSyncModel

getSyncModel

Get sync model from inp_sync object obtained by getInpSync()

Description

Get sync model from inp_sync object obtained by getInpSync()

Usage

getSyncModel()
getSyncModel

inp_sync,
silent = TRUE,
fine_tune = FALSE,
max_iter = 100,
tmb_smartsearch = TRUE
)

Arguments

inp_sync Input data prepared for the sync model using getInpSync()
silent Keep TMB quiet
fine_tune Logical. Whether to re-run the sync model excluding residual outliers. Deprecated use fineTuneSyncModel() instead.
max_iter Max number of iterations to run TMB. Default=100 seems to work in most cases.
tmb_smartsearch Logical whether to use the TMB smartsearch in the inner optimizer (see ?TMB::MakeADFun for info). Default and original implementation is TRUE. However, there seems to be an issue with some versions of Matrix that requires tmb_smartsearch=FALSE.

Value

List containing relevant data constituting the sync_model ready for use in fineTuneSyncModel() if needed or in applySync()

Examples

library(yaps)
set.seed(42)

# # # Example using the ssu1 data included in package. See ?ssu1 for info.
# # # Set parameters to use in the sync model - these will differ per study
max_epo_diff <- 120
min_hydros <- 2
time_keeper_idx <- 5
fixed_hydros_idx <- c(2:3, 6, 8, 11, 13:17)
n_offset_day <- 2
n_ss_day <- 2
keep_rate <- 20

# # # Get input data ready for getSyncModel()
inp_sync <- getInpSync(sync_dat=ssu1, max_epo_diff, min_hydros, time_keeper_idx,
fixed_hydros_idx, n_offset_day, n_ss_day, keep_rate=keep_rate, silent_check=TRUE)

# # # Check that inp_sync is ok
checkInpSync(inp_sync, silent_check=FALSE)

# # # Also take a look at coverage of the sync data
getSyncCoverage(inp_sync, plot=TRUE)
## Fit the sync model

```r
sync_model <- getSyncModel(inp_sync, silent=TRUE, max_iter=200, tmb_smartsearch = TRUE)
```

## On some systems it might work better, if we disable the smartsearch feature in TMB

## To do so, set `tmb_smartsearch = FALSE` in `getSyncModel()`

## Visualize the resulting sync model

```r
plotSyncModelResids(sync_model, by = "overall")
plotSyncModelResids(sync_model, by = "quantiles")
plotSyncModelResids(sync_model, by = "sync_tag")
plotSyncModelResids(sync_model, by = "hydro")
plotSyncModelResids(sync_model, by = "temporal_hydro")
plotSyncModelResids(sync_model, by = "temporal_sync_tag")
```

## If the above plots show outliers, `sync_model` can be fine tuned by excluding these.

## Use `fineTuneSyncModel()` for this.

## This should typically be done sequentially using `eps_thresholds` of e.g. 1E4, 1E3, 1E2, 1E2

```r
sync_model <- fineTuneSyncModel(sync_model, eps_threshold=1E3, silent=TRUE)
sync_model <- fineTuneSyncModel(sync_model, eps_threshold=1E2, silent=TRUE)
```

## Apply the `sync_model` to detections data.

```r
detections_synced <- applySync(toa=ssu1$detections, hydros=ssu1$hydros, sync_model)
```

## Prepare data for running yaps

```r
hydros_yaps <- data.table::data.table(sync_model$pl$TRUE_H)
colnames(hydros_yaps) <- c('hx', 'hy', 'hz')
focal_tag <- 15266
rbi_min <- 20
rbi_max <- 40
synced_dat <- detections_synced[tag == focal_tag]
toa <- getToaYaps(synced_dat=synced_dat, hydros=hydros_yaps, pingType='rbi',
rbi_min=rbi_min, rbi_max=rbi_max)
bbox <- getBbox(hydros_yaps, buffer=50, pen=1e6)
inp <- getInp(hydros_yaps, toa, E_dist="Mixture", n_ss=5, pingType="rbi",
            sdInits=1, rbi_min=rbi_min, rbi_max=rbi_max, ss_data_what="est", ss_data=0, bbox=bbox)
```

## Check that `inp` is ok

```r
checkInp(inp)
```

## Run yaps on the prepared data to estimate track

```r
yaps_out <- runYaps(inp, silent=TRUE, tmb_smartsearch=TRUE, maxIter=5000)
```

## Plot the results and compare to "the truth" obtained using gps

```r
oldpar <- par(no.readonly = TRUE)
par(mfrow=c(2,2))
plot(hy~hx, data=hydros_yaps, asp=1, xlab="UTM X", ylab="UTM Y", pch=20, col="green")
lines(utm_y~utm_x, data=ssu1$gps, col="blue", lwd=2)
lines(y~x, data=yaps_out$track, col="red")
points(x~top, data=yaps_out$track, col="red")
```
getToaYaps

Build TOA matrix from synced data.table - also do some pre-filtering of severe MP, pruning loose ends etc

Description

Build TOA matrix from synced data.table - also do some pre-filtering of severe MP, pruning loose ends etc

Usage

getoaYaps(synced_dat, hydros, rbi_min, rbi_max, pingType = NULL)

Arguments

synced_dat data.table containing synchronized data formatted as output from/or obtained using applySync()

hydros Dataframe from simHydros() or Dataframe with columns hx and hy containing positions of the receivers. Translate the coordinates to get the grid centre close to (0;0).

rbi_min Minimum and maximum BI for random burst interval transmitters

rbi_max Minimum and maximum BI for random burst interval transmitters

pingType Type of transmitter to simulate - either stable burst interval (‘sbi’), random burst interval (‘rbi’) or random burst interval but where the random sequence is known a priori

Value

Matrix of time-of-arrivals. One column per hydro, one row per ping.
Examples

library(yaps)
set.seed(42)

## # Example using the ssu1 data included in package. See ?ssu1 for info.
## # Set parameters to use in the sync model - these will differ per study
max_epo_diff <- 120
min_hydros <- 2
time_keeper_idx <- 5
fixed_hydros_idx <- c(2:3, 6, 8, 11, 13:17)
n_offset_day <- 2
n_ss_day <- 2
keep_rate <- 20

## # Get input data ready for getSyncModel()
inp_sync <- getInpSync(sync_dat=ssu1, max_epo_diff, min_hydros, time_keeper_idx,
                      fixed_hydros_idx, n_offset_day, n_ss_day, keep_rate=keep_rate, silent_check=TRUE)

## # Check that inp_sync is ok
checkInpSync(inp_sync, silent_check=FALSE)

## # Also take a look at coverage of the sync data
getSyncCoverage(inp_sync, plot=TRUE)

## # Fit the sync model
sync_model <- getSyncModel(inp_sync, silent=TRUE, max_iter=200, tmb_smartsearch = TRUE)

## # On some systems it might work better, if we disable the smartsearch feature in TMB
## # To do so, set tmb_smartsearch = FALSE in getSyncModel()

## # Visualize the resulting sync model
plotSyncModelResids(sync_model, by = "overall")
plotSyncModelResids(sync_model, by = "quantiles")
plotSyncModelResids(sync_model, by = "sync_tag")
plotSyncModelResids(sync_model, by = "hydro")
plotSyncModelResids(sync_model, by = "temporal_hydro")
plotSyncModelResids(sync_model, by = "temporal_sync_tag")

## # If the above plots show outliers, sync_model can be fine tuned by excluding these.
## # Use fineTuneSyncModel() for this.
## # This should typically be done sequentially using eps_thresholds of e.g. 1E4, 1E3, 1E2, 1E2
sync_model <- fineTuneSyncModel(sync_model, eps_threshold=1E3, silent=TRUE)
sync_model <- fineTuneSyncModel(sync_model, eps_threshold=1E2, silent=TRUE)

## # Apply the sync_model to detections data.
detections_synced <- applySync(toa=ssu1$detections, hydros=ssu1$hydros, sync_model)

## # Prepare data for running yaps
hydros_yaps <- data.table::data.table(sync_model$pl$TRUE_H)
colnames(hydros_yaps) <- c('hx','hy','hz')
focal_tag <- 15266
rbi_min <- 20
rbi_max <- 40
synced_dat <- detections_synced[tag == focal_tag]
toa <- getToaYaps(synced_dat=synced_dat, hydros=hydros_yaps, pingType='rbi',
rbi_min=rbi_min, rbi_max=rbi_max)
bbox <- getBbox(hydros_yaps, buffer=50, pen=1e6)
inp <- getInp(hydros_yaps, toa, E_dist="Mixture", n_ss=5, pingType="rbi",
ssInits=1, rbi_min=rbi_min, rbi_max=rbi_max, ss_data_what="est", ss_data=0, bbox=bbox)

## Check that inp is ok
checkInp(inp)

## Run yaps on the prepared data to estimate track
yaps_out <- runYaps(inp, silent=TRUE, tmb_smartsearch=TRUE, maxIter=5000)

## Plot the results and compare to "the truth" obtained using gps
oldpar <- par(no.readonly = TRUE)
par(mfrow=c(2,2))
plot(hy~hx, data=hydros_yaps, asp=1, xlab="UTM X", ylab="UTM Y", pch=20, col="green")
lines(utm_y~utm_x, data=ssu1$gps, col="blue", lwd=2)
lines(y~x, data=yaps_out$track, col="red")

plot(utm_x~ts, data=ssu1$gps, col="blue", type="l", lwd=2)
points(x~top, data=yaps_out$track, col="red")
lines(x~top, data=yaps_out$track, col="red")
lines(x+2*x_sd~top, data=yaps_out$track, col="red", lty=2)
lines(x-2*x_sd~top, data=yaps_out$track, col="red", lty=2)

plot(utm_y~ts, data=ssu1$gps, col="blue", type="l", lwd=2)
points(y~top, data=yaps_out$track, col="red")
lines(y~top, data=yaps_out$track, col="red")
lines(y+2*y_sd~top, data=yaps_out$track, col="red", lty=2)
lines(y-2*y_sd~top, data=yaps_out$track, col="red", lty=2)

plot(nobs~top, data=yaps_out$track, type="p", main="#detecting hydros per ping")
lines(caTools::runmean(nobs, k=10)+top, data=yaps_out$track, col="orange", lwd=2)
par(oldpar)

---

**plotBbox**

Graphical representation of spatial constraints

**Description**

Graphical representation of spatial constraints

**Usage**

```r
plotBbox(hydros, bbox)
```
Arguments

hydros  
Dataframe from simHydros() or Dataframe with columns hx and hy containing positions of the receivers. Translate the coordinates to get the grid centre close to (0;0).

bbox  
Spatial constraints in the form of a bounding box. See ?getBbox for details.

Value

No return value, called to plot graphic.

Examples

```r
hydros <- ssu1$hydros
colnames(hydros) <- c('serial','hx','hy','hz','sync_tag','idx')
bbox <- getBbox(hydros)
plotBbox(hydros, bbox)
```

Description

Delta values indicate absolute difference between true and estimated distances based on pairwise relative distances to sync_tag. For instance, a ping from sync_tag t colocated with hydro Ht is detected by hydros H1 and H2. The pairwise relative distance to sync tag is then delta = abs((true_dist(Ht, H1) - true_dist(Ht, H2)) - (est_dist(Ht, H1) - est_dist(Ht, H2))

Usage

```r
plotSyncModelCheck(sync_model, by = "")
```

Arguments

sync_model  
Synchronization model obtained using getSyncModel()

by  
What to facet/group the plot by? Currently supports one of 'sync_bin_sync', 'sync_bin_hydro', 'sync_bin_sync_smooth', 'syncbinhydro_smooth', 'hydro', 'sync_tag'

Value

No return value, called to plot graphics.
Examples

```
sync_model <- ssu1$sync_model
plotSyncModelHydros(sync_model)

plotSyncModelResids(sync_model, by = "overall")
plotSyncModelResids(sync_model, by = "quantiles")
plotSyncModelResids(sync_model, by = "sync_tag")
plotSyncModelResids(sync_model, by = "hydro")
plotSyncModelResids(sync_model, by = "temporal")
plotSyncModelResids(sync_model, by = "temporal_hydro")
plotSyncModelResids(sync_model, by = "temporal_sync_tag")
plotSyncModelCheck(sync_model, by = "hydro")
plotSyncModelCheck(sync_model, by = "sync_tag")
plotSyncModelCheck(sync_model, by = "sync_bin_sync")
plotSyncModelCheck(sync_model, by = "sync_bin_hydro")
```

Description

Plot hydrophone positions. Especially useful if some hydro re-positioned as part of the sync model.

Usage

```
plotSyncModelHydros(sync_model)
```

Arguments

- `sync_model`: Synchronization model obtained using `getSyncModel()`

Value

No return value, called to plot graphics.

Examples

```
sync_model <- ssu1$sync_model
plotSyncModelHydros(sync_model)

plotSyncModelResids(sync_model, by = "overall")
plotSyncModelResids(sync_model, by = "quantiles")
```
The function `plotSyncModelResids` is designed to plot residuals of a synchronization model to enable checking of the model. It takes a synchronization model (`sync_model`) as input and optionally groups the residuals by different factors such as `sync_tag`, `hydro`, `quantiles`, `temporal`, `temporal_hydro`, or `temporal_sync_tag`.

**Usage**

```r
plotSyncModelResids(sync_model, by = "overall")
```

**Arguments**

- `sync_model` (required): Synchronization model obtained using `getSyncModel()`.

**Description**

Plot residuals of `sync_model` to enable check of model.

**Examples**

```r
sync_model <- ssul$sync_model

plotSyncModelHydros(sync_model)
plot SyncModelResids(sync_model, by = "overall")
plot SyncModelResids(sync_model, by = "quantiles")
plot SyncModelResids(sync_model, by = "sync_tag")
plot SyncModelResids(sync_model, by = "hydro")
plot SyncModelResids(sync_model, by = "temporal")
plot SyncModelResids(sync_model, by = "temporal_hydro")
plot SyncModelResids(sync_model, by = "temporal_sync_tag")
plot SyncModelCheck(sync_model, by = "hydro")
```
**plotYaps**

Basic plots of yaps output

**Usage**

```
plotYaps(yaps_out, type = "map", xlim = NULL, ylim = NULL, main = NULL)
```

**Arguments**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>yaps_out</td>
<td>Output from successful run of runYaps()</td>
</tr>
<tr>
<td>type</td>
<td>Plot type. type=&quot;map&quot; produces a basic map of estimated track and hydrophones; type=&quot;coord_X&quot;, type=&quot;coord_Y&quot; produces plots of X and Y coordinated including + 1 standard error.</td>
</tr>
<tr>
<td>xlim, ylim</td>
<td>Optional vectors of length 2 to set xlim and/or ylim.</td>
</tr>
<tr>
<td>main</td>
<td>Title of plot - optional.</td>
</tr>
</tbody>
</table>

**Value**

No return value, called to plot graphics.

**Examples**

```
library(yaps)
plotYaps(ssu1$yaps_out, type = "map")
plotYaps(ssu1$yaps_out, type = "coord_X")
plotYaps(ssu1$yaps_out, type = "coord_Y")
```
### prepDetections

Experimental! Prepare detections data.table from raw data - csv-files exported from vendor software

#### Description

Experimental! Prepare detections data.table from raw data - csv-files exported from vendor software

#### Usage

`prepDetections(raw_dat, type)`

#### Arguments

- `raw_dat`: Data file from vendor supplied software
- `type`: Type of the vendor file. Currently only ’vemco_vue’ is supported.

#### Value

data.table containing detections extracted from manufacturer data file.

#### Examples

```r
## Not run:
prepped_detections <- prepDetections("path-to-raw-data-file", type="vemco_vue")
## End(Not run)
```

### runYaps

Function to run TMB to estimate track

#### Description

Function to run TMB to estimate track

#### Usage

`runYaps(
inp,
maxIter = 1000,
getPlsd = TRUE,
getRep = TRUE,
silent = TRUE,
opt_fun = "nlminb",
opt_controls = list(),
tmb_smartsearch = TRUE
)`
```r
runTmb(
  inp, 
  maxIter = 1000, 
  getPlsd = TRUE, 
  getRep = TRUE, 
  silent = TRUE, 
  opt_fun = "nlminb", 
  opt_controls = list(), 
  tmb_smartsearch = TRUE
)
```

**Arguments**

- `inp`: inp-object obtained from `getInp()`
- `maxIter`: Sets `inner.control(maxit)` of the TMB-call. Increase if model is not converging.
- `getPlsd`, `getRep`: Whether or not to get sd estimates (plsd=TRUE) and reported values (getRep=TRUE).
- `silent`: Logical whether to keep the optimization quiet.
- `opt_fun`: Which optimization function to use. Default is `opt_fun = 'nlminb'` - alternative is `opt_fun = 'nloptr'` (experimental!). If using `nloptr`, `opt_controls` must be specified.
- `opt_controls`: List of controls passed to optimization function. For instances, tolerances such as `xtol_abs=1e-12,maxeval=2E+4,print_level = 1,local_opts= list(algorithm="NLOPT_LD_AUGLAG_EQ",xtol_rel=1e-4)`. See `?nloptr` and the NLopt site https://nlopt.readthedocs.io/en/latest/ for more info. Some algorithms in `nloptr` require bounded parameters - this is not currently implemented.
- `tmb_smartsearch`: Logical whether to use the TMB smartsearch in the inner optimizer (see `?TMB::MakeADFun` for info). Default and original implementation is TRUE. However, there seems to be an issue with recent versions of `Matrix` that requires `tmb_smartsearch=FALSE`.

**Value**

List containing results of fitting `yaps` to the data.

- `pl`: List containing all parameter estimates.
- `plsd`: List containing standard errors of parameter estimates.
- `rep`: List containing `mu_toa`.
- `obj`: Numeric obj value of the fitted model obtained using `obj$fn()`.
**inp** List containing the inp object used in `runYaps()`. See `getInp` for further info.

**conv_status** Integer convergence status.

**conv_message** Text version of convergence status.

**track** A data.table containing the estimated track including time-of-ping (top), standard errors and number of hydros detecting each ping (nobs).

### Examples

```r
library(yaps)
set.seed(42)

# # # Example using the ssu1 data included in package. See ?ssu1 for info.
# # # Set parameters to use in the sync model - these will differ per study
max_epo_diff <- 120
min_hydros <- 2
time_keeper_idx <- 5
fixed_hydros_idx <- c(2:3, 6, 8, 11, 13:17)
n_offset_day <- 2
n_ss_day <- 2
keep_rate <- 20

# # # Get input data ready for `getSyncModel()`
inp_sync <- getInpSync(sync_dat=ssu1, max_epo_diff, min_hydros, time_keeper_idx,
fixed_hydros_idx, n_offset_day, n_ss_day, keep_rate=keep_rate, silent_check=TRUE)

# # # Check that inp_sync is ok
checkInpSync(inp_sync, silent_check=FALSE)

# # # Also take a look at coverage of the sync data
getSyncCoverage(inp_sync, plot=TRUE)

# # # Fit the sync model
sync_model <- getSyncModel(inp_sync, silent=TRUE, max_iter=200, tmb_smartsearch = TRUE)

# # # On some systems it might work better, if we disable the smartsearch feature in TMB
# # # To do so, set tmb_smartsearch = FALSE in getSyncModel()

# # # Visualize the resulting sync model
plotSyncModelResids(sync_model, by = "overall")
plotSyncModelResids(sync_model, by = "quantiles")
plotSyncModelResids(sync_model, by = "sync_tag")
plotSyncModelResids(sync_model, by = "hydro")
plotSyncModelResids(sync_model, by = "temporal_hydro")
plotSyncModelResids(sync_model, by = "temporal_sync_tag")

# # # If the above plots show outliers, sync_model can be fine tuned by excluding these.
# # # Use `fineTuneSyncModel()` for this.
# # # This should typically be done sequentially using eps_thresholds of e.g. 1E4, 1E3, 1E2, 1E2
sync_model <- fineTuneSyncModel(sync_model, eps_threshold=1E3, silent=TRUE)
sync_model <- fineTuneSyncModel(sync_model, eps_threshold=1E2, silent=TRUE)
```
## Apply the sync_model to detections data.

detections_synced <- applySync(toa=ssu1$detections, hydros=ssu1$hydros, sync_model)

## Prepare data for running yaps

hydros_yaps <- data.table::data.table(sync_model$pl$TRUE_H)
colnames(hydros_yaps) <- c('hx', 'hy', 'hz')
focal_tag <- 15266
rbi_min <- 20
rbi_max <- 40
synced_dat <- detections_synced[tag == focal_tag]
toa <- getToaYaps(synced_dat=synced_dat, hydros=hydros_yaps, pingType='rbi',
    rbi_min=rbi_min, rbi_max=rbi_max)
bbox <- getBbox(hydros_yaps, buffer=50, pen=1e6)
inp <- getInp(hydros_yaps, toa, E_dist="Mixture", n_ss=5, pingType="rbi",
    sdInits=1, rbi_min=rbi_min, rbi_max=rbi_max, ss_data_what="est", ss_data=0, bbox=bbox)

## Check that inp is ok

checkInp(inp)

## Run yaps on the prepared data to estimate track

yaps_out <- runYaps(inp, silent=TRUE, tmb_smartsearch=TRUE, maxIter=5000)

## Plot the results and compare to "the truth" obtained using gps

oldpar <- par(no.readonly = TRUE)
par(mfrow=c(2,2))
plot(hy~hx, data=hydros_yaps, asp=1, xlab="UTM X", ylab="UTM Y", pch=20, col="green")
lines(utm_y~utm_x, data=ssu1$gps, col="blue", lwd=2)
lines(y~x, data=yaps_out$track, col="red")

plot(utm_x~ts, data=ssu1$gps, col="blue", type="l", lwd=2)
points(x~top, data=yaps_out$track, col="red")
lines(x~2*x_sd~top, data=yaps_out$track, col="red", lty=2)
lines(x+2*x_sd~top, data=yaps_out$track, col="red", lty=2)

plot(utm_y~ts, data=ssu1$gps, col="blue", type="l", lwd=2)
points(y~top, data=yaps_out$track, col="red")
lines(y~2*y_sd~top, data=yaps_out$track, col="red", lty=2)
lines(y+2*y_sd~top, data=yaps_out$track, col="red", lty=2)

plot(nobs~top, data=yaps_out$track, type="p", main="#detecting hydros per ping")
lines(caTools::runmean(nobs, k=10)-top, data=yaps_out$track, col="orange", lwd=2)
par(oldpar)
**Description**

Sim hydrophone array configuration

**Usage**

```r
simHydros(auto = TRUE, trueTrack = NULL)
```

**Arguments**

- `auto` If TRUE, attempts to find a decent array configuration to cover the simulated true track.
- `trueTrack` Track obtained from `simTrueTrack()`.

**Value**

data.frame containing X and Y for hydros

**Examples**

```r
library(yaps)
set.seed(42)
# Simulate true track of animal movement of n seconds
trueTrack <- simTrueTrack(model="crw", n = 1000, deltaTime=1, shape=1, scale=0.5, addDielPattern=TRUE, ss='rw')
# Simulate telemetry observations from true track.
# Format and parameters depend on type of transmitter burst interval (BI).
pingType <- 'sbi'
if(pingType == 'sbi') { # stable BI
    sbi_mean <- 30; sbi_sd <- 1e-4;
    teleTrack <- simTelemetryTrack(trueTrack, pingType=pingType, sbi_mean=sbi_mean, sbi_sd=sbi_sd)
} else if(pingType == 'rbi') { # random BI
    pingType <- 'rbi'; rbi_min <- 20; rbi_max <- 40;
    teleTrack <- simTelemetryTrack(trueTrack, pingType=pingType, rbi_min=rbi_min, rbi_max=rbi_max)
}
# Simulate hydrophone array
hydros <- simHydros(auto=TRUE, trueTrack=trueTrack)
toa_list <- simToa(teleTrack, hydros, pingType, sigmaToa=1e-4, pNA=0.25, pMP=0.01)
toa <- toa_list$toa
# Specify whether to use ss_data from measured water temperature (ss_data$what <- 'data') or
# to estimate ss in the model (ss_data$what <- 'est')
ss_data$what <- 'data'
if(ss_data$what == 'data') {ss_data <- teleTrack$ss} else {ss_data <- 0}
if(pingType == 'sbi'){
    inp <- getInp(hydros, toa, E_dist="Mixture", n_ss=10, pingType=pingType, sdInits=0,
```
Based on a known true track obtained using simTrueTrack, this function will give true positions at time-of-pings, which are also in the output. TOPs are determined by user-specified transmitter type. Number of pings are determined automatically based on track length and transmitter specifications.

Usage

```r
simTelemetryTrack(
  trueTrack,
  pingType,
  sbi_mean = NULL,
  sbi_sd = NULL,
  rbi_min = NULL,
  rbi_max = NULL
)
```
simTelemetryTrack

Arguments

- **trueTrack**: Know track obtained using simTrueTrack
- **pingType**: Type of transmitter to simulate - either stable burst interval ('sbi'), random burst interval ('rbi') or random burst interval but where the random sequence is known a priori
- **sbi_mean, sbi_sd**: Mean and SD of burst interval when pingType = 'sbi'
- **rbi_min, rbi_max**: Minimum and maximum BI for random burst interval transmitters

Value

data.frame containing time of ping and true positions

Examples

```r
library(yaps)
set.seed(42)
# Simulate true track of animal movement of n seconds
trueTrack <- simTrueTrack(model="crw", n = 1000, deltaTime=1, shape=1, scale=0.5, addDielPattern=TRUE, ss="rw")

# Simulate telemetry observations from true track.
# Format and parameters depend on type of transmitter burst interval (BI).
pingType <- 'sbi'

if(pingType == 'sbi') { # stable BI
  sbi_mean <- 30; sbi_sd <- 1e-4;
  teleTrack <- simTelemetryTrack(trueTrack, pingType=pingType, sbi_mean=sbi_mean, sbi_sd=sbi_sd)
} else if(pingType == 'rbi'){ # random BI
  pingType <- 'rbi'; rbi_min <- 20; rbi_max <- 40;
  teleTrack <- simTelemetryTrack(trueTrack, pingType=pingType, rbi_min=rbi_min, rbi_max=rbi_max)
}

# Simulate hydrophone array
hydros <- simHydros(auto=TRUE, trueTrack=trueTrack)
toa_list <- simToa(teleTrack, hydros, pingType, sigmaToa=1e-4, pNA=0.25, pMP=0.01)
toa <- toa_list$toa

# Specify whether to use ss_data from measured water temperature (ss_data_what <- 'data') or
# to estimate ss in the model (ss_data_what <- 'est')
ss_data_what <- 'data'
if(ss_data_what == 'data') {ss_data <- teleTrack$ss} else {ss_data <- 0}

if(pingType == 'sbi'){
  inp <- getInp(hydros, toa, E_dist="Mixture", n_ss=10, pingType=pingType, sdInits=0, ss_data_what=ss_data_what, ss_data=ss_data)
} else if(pingType == 'rbi'){
```

inp <- getInp(hydros, toa, E_dist="Mixture", n_ss=10, pingType=pingType, sdInits=0, rbi_min=rbi_min, rbi_max=rbi_max, ss_data_what=ss_data_what, ss_data=ss_data)

dpl <- c()
maxIter <- ifelse(pingType=="sbi", 500, 5000)
outTmb <- runYaps(inp, maxIter=maxIter, getPlsd=TRUE, getRep=TRUE)

# Estimates in pl
pl <- outTmb$pl
# Correcting for hydrophone centering
pl$X <- outTmb$pl$X + inp$inp_params$Hx0
pl$Y <- outTmb$pl$Y + inp$inp_params$Hy0

# Error estimates in plsd
plsd <- outTmb$plsd

# plot the resulting estimated track
plot(y~x, data=trueTrack, type="l", xlim=range(hydros$hx), ylim=range(hydros$hy), asp=1)
lines(y~x, data=teleTrack)
points(hy~hx, data=hydros, col="green", pch=20, cex=3)
lines(pl$Y~pl$X, col="red")

simToa

Sim TOA matrix for the supplied telemetryTrack

Description
Provides the TOA matrix for the specified telemetryTrack. Probability of NA (pNA) and observation noise (sigmaToa) can be specified.

Usage
simToa(telemetryTrack, hydros, pingType, sigmaToa, pNA, pMP, tempRes = NA)

Arguments

telemetryTrack  Dataframe obtained from simTelemetryTrack
hydros  Dataframe obtained from getHydros
pingType  Type of transmitter to simulate - either stable burst interval (\'sbi\'), random burst interval (\'rbi\') or random burst interval but where the random sequence is known a priori
sigmaToa  Detection uncertainty
pNA  Probability of missing detection 0-1
pMP  Probability of multipath propagated signal 0-1
tempRes  Temporal resolution of the hydrophone. PPM systems are typically 1/1000 sec. Other systems are as high as 1/19200 sec.
Value

List containing TOA matrix (toa) and matrix indicating, which obs are multipath (mp_mat)

Examples

```r
library(yaps)
set.seed(42)
# Simulate true track of animal movement of n seconds
trueTrack <- simTrueTrack(model="crw", n = 1000, deltaTime=1, shape=1, scale=0.5, addDielPattern=TRUE, ss="rw")

# Simulate telemetry observations from true track.
# Format and parameters depend on type of transmitter burst interval (BI).
pingType <- 'sbi'
if(pingType == 'sbi') { # stable BI
  sbi_mean <- 30; sbi_sd <- 1e-4;
  teleTrack <- simTelemetryTrack(trueTrack, pingType=pingType, sbi_mean=sbi_mean, sbi_sd=sbi_sd)
} else if(pingType == 'rbi') { # random BI
  pingType <- 'rbi'; rbi_min <- 20; rbi_max <- 40;
  teleTrack <- simTelemetryTrack(trueTrack, pingType=pingType, rbi_min=rbi_min, rbi_max=rbi_max)
}

# Simulate hydrophone array
hydros <- simHydros(auto=TRUE, trueTrack=trueTrack)
toa_list <- simToa(teleTrack, hydros, pingType, sigmaToa=1e-4, pNA=0.25, pMP=0.01)
toa <- toa_list$toa

# Specify whether to use ss_data from measured water temperature (ss_data_what <- 'data') or
# to estimate ss in the model (ss_data_what <- 'est')
ss_data_what <- 'data'
if(ss_data_what == 'data') {ss_data <- teleTrack$ss} else {ss_data <- 0}

if(pingType == 'sbi'){
  inp <- getInp(hydros, toa, E_dist="Mixture", n_ss=10, pingType=pingType, sdInits=0,
               ss_data_what=ss_data_what, ss_data=ss_data)
} else if(pingType == 'rbi'){
  inp <- getInp(hydros, toa, E_dist="Mixture", n_ss=10, pingType=pingType, sdInits=0,
               rbi_min=rbi_min, rbi_max=rbi_max, ss_data_what=ss_data_what, ss_data=ss_data)
}

pl <- c()
maxIter <- ifelse(pingType=="sbi", 500, 5000)
outTmb <- runYaps(inp, maxIter=maxIter, getPlsd=TRUE, getRep=TRUE)

# Estimates in pl
pl <- outTmb$pl
# Correcting for hydrophone centering
pl$X <- outTmb$pl$X + inp$inp_params$Hx0
pl$Y <- outTmb$pl$Y + inp$inp_params$Hy0
```
# Error estimates in plsd
plsd <- outTmb$plsd

# plot the resulting estimated track
plot(y=x, data=trueTrack, type="l", xlim=range(hydros$hx), ylim=range(hydros$hy), asp=1)
lines(y=x, data=teleTrack)
points(hy~hx, data=hydros, col="green", pch=20, cex=3)
lines(pl$Y~pl$X, col="red")

---

**simTrueTrack**

*Simulate a known movement track for subsequent estimation using YAPS*

**Description**

Produces a simulated regular time-spaced track following the specified movement model. Linear movement between consecutive observations is assumed. The output contains x, y, time and sound speed at each simulated position.

**Usage**

```r
simTrueTrack(
  model = "rw",
  n,
  deltaTime = 1,
  D = NULL,
  shape = NULL,
  scale = NULL,
  addDielPattern = TRUE,
  ss = "rw",
  start_pos = NULL
)
```

**Arguments**

- **model**: Movement model: 'rw': Two-dimension random walk (X,Y)
- **n**: Number of steps in the simulated track
- **deltaTime**: Number of time units (seconds) between each location
- **D**: Diffusivity of the animal movement - only used if model='rw'
- **shape**: Shape of the Weibull distribution - only used when model=’crw’.
- **scale**: Scale of the Weibull distribution - only used when model=’crw’.
- **addDielPattern**: Adds a realistic(?) diel pattern to movement. Periods of both low and high movement
- **ss**: Simulations model for Speed of Sound - defaults to 'rw' = RW-model.
- **start_pos**: Specify the starting position of the track with c(x0, y0)
Value
data.frame containing a simulated track

Examples

library(yaps)
set.seed(42)
# Simulate true track of animal movement of n seconds
trueTrack <- simTrueTrack(model='crw', n = 1000, deltaTime=1, shape=1,
scale=0.5, addDielPattern=TRUE, ss='rw')

# Simulate telemetry observations from true track.
# Format and parameters depend on type of transmitter burst interval (BI).
pingType <- 'sbi'

if(pingType == 'sbi') { # stable BI
  sbi_mean <- 30; sbi_sd <- 1e-4;
teleTrack <- simTelemetryTrack(trueTrack, pingType=pingType, sbi_mean=sbi_mean, sbi_sd=sbi_sd)
} else if(pingType == 'rbi'){ # random BI
  pingType <- 'rbi'; rbi_min <- 20; rbi_max <- 40;
teleTrack <- simTelemetryTrack(trueTrack, pingType=pingType, rbi_min=rbi_min, rbi_max=rbi_max)
}

# Simulate hydrophone array
hydros <- simHydros(auto=TRUE, trueTrack=trueTrack)
toa_list <- simToa(teleTrack, hydros, pingType, sigmaToa=1e-4, pNA=0.25, pMP=0.01)
toa <- toa_list$toa

# Specify whether to use ss_data from measured water temperature (ss_data_what <- 'data') or
# to estimate ss in the model (ss_data_what <- 'est')
ss_data_what <- 'data'
if(ss_data_what == 'data') {ss_data <- teleTrack$ss} else {ss_data <- 0}

if(pingType == 'sbi'){
  inp <- getInp(hydros, toa, E_dist="Mixture", n_ss=10, pingType=pingType, sdInits=0,
  ss_data_what=ss_data_what, ss_data=ss_data)
} else if(pingType == 'rbi'){#
  inp <- getInp(hydros, toa, E_dist="Mixture", n_ss=10, pingType=pingType, sdInits=0,
  rbi_min=rbi_min, rbi_max=rbi_max, ss_data_what=ss_data_what, ss_data=ss_data)
}

pl <- c()
maxIter <- ifelse(pingType=="sbi", 500, 5000)
outTmb <- runYaps(inp, maxIter=maxIter, getPlsd=TRUE, getRep=TRUE)

# Estimates in pl
pl <- outTmb$pl
# Correcting for hydrophone centering
pl$X <- outTmb$pl$X + inp$inp_params$Hx0
pl$Y <- outTmb$pl$Y + inp$inp_params$Hy0
# Error estimates in plsd
plsd <- outTmb$plsd

# plot the resulting estimated track
plot(y~x, data=trueTrack, type="l", xlim=range(hydros$hx), ylim=range(hydros$hy), asp=1)
lines(y~x, data=teleTrack)
points(hy~hx, data=hydros, col="green", pch=20, cex=3)
lines(pl$Y~pl$X, col="red")

---

**ssu1**

**Test data from Florida Bay**

**Description**

Small data set collected for positioning using acoustic telemetry and YAPS. The data are part of a feasibility study using YAPS on Vemco PPM style data to track fish in shallow parts of Florida Bay. Data were collected using VR2 (Vemco) hydrophones. Included in yaps with permission from J.S. Rehage, FIU Florida International University.

**Usage**

ssu1

**Format**

A list containing 3 data.tables:

- **hydros**
  - serial Hydrophone serial number.
  - x,y,z Position of hydrophones in UTM.
  - sync_tag ID of co-located sync tag. Must be identical to entries in data.table detections$tag.
  - idx Unique values from 1:nrow(hydros).

- **detections**
  - ts Timestamp of detection in POSIXct().
  - tag ID of detected tag.
  - epo Timestamp as number of seconds since Unix epoch. Can be obtained using as.numeric(ts).
  - frac Sub-second part of detection timestamp in fractions of second (0-1).
  - serial Serial number of detecting hydrophone. Must match entry in data.table hydros.

- **gps**
  - ts Timestamp of gps position in POSIXct().
  - utm_x, utm_y Coordinates of position. Same projection and coordinate system as used in hydros.
tempToSs  
*Calculate speed of sound from water temperature, salinity and depth*
*Based on H. Medwin (1975) Speed of sound in water: A simple equation for realistic parameters. (https://doi.org/10.1121/1.380790)*

**Description**

Calculate speed of sound from water temperature, salinity and depth Based on H. Medwin (1975) Speed of sound in water: A simple equation for realistic parameters. (https://doi.org/10.1121/1.380790)

**Usage**

```
tempToSs(temp, sal, depth = 5)
```

**Arguments**

- **temp**: Water temperature in degrees Celsius
- **sal**: Water salinity in parts per thousand (promille)
- **depth**: Depth in meters - default = 5 m - can typically be ignored

**Value**

Vector of estimated speed of sound in water.

**Examples**

```r
water_temp <- rnorm(100, 20, 2)
s <- tempToSs(temp=water_temp, sal=0, depth=5)
```

---

testYaps  
*Test YAPS core functionality*

**Description**

Run `testYaps()` to check that the core functions of YAPS is working correctly. Output should be a random simulated (black) and estimated (red) track.

**Usage**

```
testYaps(
silent = TRUE,
pingType = "sbi",
est_ss = TRUE,
opt_fun = "nlminb",
opt_controls = list(),
return_yaps = FALSE,
tmb_smartsearch = TRUE
)
```
Arguments

- **silent**
  Logical whether to print output to the console

- **pingType**
  Type of transmitter to simulate - either stable burst interval (‘sbi’), random burst interval (‘rbi’) or random burst interval but where the random sequence is known a priori

- **est_ss**
  Logical whether to test using ss_data what = 'est' (est_ss = TRUE) or ss_data what = 'data' (est_ss = FALSE)

- **opt_fun**
  Which optimization function to use. Default is opt_fun = 'nlminb' - alternative is opt_fun = 'nloptr' (experimental!). If using nloptr, opt_controls must be specified.

- **opt_controls**
  List of controls passed to optimization function. For instances, tolerances such as x.tol=1E-8.
  If opt_fun = 'nloptr', opt_controls must be a list formatted appropriately.
  For instance:

  ```r
  opt_controls <- list(
    algorithm="NLOPT_LD_AUGLAG",
    xtol_abs=1e-12, maxeval=2E+4, print_level = 1, local_opts=list(
      algorithm="NLOPT_LD_AUGLAG_EQ", xtol_rel=1e-4
    )
  )
  ```

  See ?nloptr and the NLopt site https://nlopt.readthedocs.io/en/latest/ for more info. Some algorithms in nloptr require bounded parameters - this is not currently implemented.

- **return_yaps**
  Logical whether to return the fitted yaps model. Default=FALSE.

- **tmb_smartsearch**
  Logical whether to use the TMB smartsearch in the inner optimizer (see ?TMB::MakeADFun for info). Default and original implementation is TRUE. However, there seems to be an issue with recent versions of Matrix that requires tmb_smartsearch=FALSE.

Value

If return_yaps == TRUE, the fitted yaps object. See ?runYaps for further info.

Examples

```r
# To test basic functionality of yaps using simulated data

testYaps()

# Three pingTypes are available:
# fixed burst interval (testYaps(pingType='sbi')),
# random burst interval with UNKNOWN burst interval sequence (testYaps(pingType='rbi')),
# random burst interval with KNOWN burst interval sequence (testYaps(pingType='pbi'))
```
Index

* datasets
  dat_align, 10
  ssu1, 43

alignBurstSeq, 2
applySync, 3

checkInp, 6
checkInpSync, 8

dat_align, 10
fineTuneSyncModel, 11

getBbox, 13
getInp, 14
getInpSync, 17
getSyncCoverage, 20
getSyncModel, 22
getToaYaps, 25

plotBbox, 27
plotSyncModelCheck, 28
plotSyncModelHydros, 29
plotSyncModelResids, 30
plotYaps, 31
prepDetections, 32

runTmb (runYaps), 32
runYaps, 32

simHydros, 35
simTelemetryTrack, 37
simToa, 39
simTrueTrack, 41
ssu1, 43

tempToSs, 44
testYaps, 44