Package 'animalEKF'

September 29, 2023

Type Package

Title Extended Kalman Filters for Animal Movement

Version 1.2

Date 2023-09-09

Author Samuel Ackerman

Maintainer Samuel Ackerman <smackrmn@gmail.com>

Description Synthetic generation of 1-D and 2-D correlated random walks (CRWs) for animal movement with behavioral switching, and particle filter estimation of movement parameters from observed trajectories using Extended Kalman Filter (EKF) model. See Ackerman (2018) <https://digital.library.temple.edu/digital/collection/p245801coll10/id/499150>.

Depends R (>= 3.3.0), shiny, sp, sf

License GPL (>= 2)

Imports MCMCpack, ellipse, mvtnorm, deldir, colorspace, Matrix, MASS, png, grDevices, bezier, HDInterval, plyr, stats, methods, utils, ggplot2, rlang

NeedsCompilation no

Repository CRAN

Date/Publication 2023-09-29 15:32:41 UTC

R topics documented:

`animalEKF-package` *Extended Kalman Filters for Animal Movement*

Description

Synthetic generation of 1-D and 2-D correlated random walks (CRWs) for animal movement with behavioral switching, and particle filter estimation of movement parameters from observed trajectories using Extended Kalman Filter (EKF) model. See Ackerman (2018) `https://digital.library.temple.edu/digital/collection/p245801coll10/id/499150`.

Note

I am indebted to Dr. Mario Espinoza and co-authors for providing the shark observation data that inspired this work, and for allowing it to be included in this package. I am especially grateful to Dr. Espinoza for many discussions regarding the nuances of modeling animal movement, particularly for suggesting the idea of behavioral switching models.

I am grateful to my doctoral thesis advisors, Dr. Marc Sobel, Dr. Richard Heiberger, and Dr. Mike O'Connor for supervising my research in this topic. I am particularly grateful to Dr. Heiberger for his many hours in advising me on package design and on the `shiny` simulations in this package.

References

Ackerman, Samuel. "A Probabilistic Characterization of Shark Movement Using Location Tracking Data." Temple University doctoral thesis, 2018. `https://digital.library.temple.edu/digital/collection/p245801coll10/id/499150`

Carvalho, Carlos M., Johannes, Michael S., Lopes, Hedibert F., and Nicholas G. Polson. "Particle Learning and Smoothing." Statistical Science, 2010.

Espinoza, Mario, Farrugia, Thomas J., Lowe, Christopher G. "Habitat Use, Movements and Site Fidelity of the Gray Smooth-Hound Shark in a Newly Restored Southern California Estuary." Journal of Experimental Marine Biology and Ecology, 2011.

bc_longlat_map *Image of Bolsa Chica for use with* shark_vis_longlat

Description

Image of Bolsa Chica for use with shark_vis_longlat

Usage

```
data(bc_longlat_map)
```

Format

The format is: List of 8 $ lat.center: Named num 33.7 ..- attr(*, "names")= chr "lat" $ lon.center: Named num -118 ..- attr(*, "names")= chr "lon" $ zoom : num 15 $ myTile : num [1:640, 1:640, 1:4] 0.639 0.639 0.639 0.639 0.639 ... $ BBOX :List of 2 ..$ ll: num [1, 1:2] 33.7 -118.1 .. ..- attr(*, "dimnames")=List of 2 .. .. ..$: NULL .. .. ..$: chr [1:2] "lat" "lon" ..$ ur: num [1, 1:2] 33.7 -118 .. ..- attr(*, "dimnames")=List of 2 .. .. ..$: NULL .. .. ..$: chr [1:2] "lat" "lon" $ url : chr "google" $ size : num [1:2] 640 640 $ SCALE : num 1 - attr(*, "class")= chr "staticMap"

Source

Google Maps.

bc_longlat_map_img_ras *Raster image of Bolsa Chica for use with* shark_vis_longlat

Description

Raster image of Bolsa Chica for use with shark_vis_longlat

Usage

```
data("bc_longlat_map_img_ras")
```

Format

The format is: 'raster' chr [1:640, 1:640] "#A3CCFFFF" "#A3CCFFFF" "#A3CCFFFF" ...

Source

Google Maps.

Examples

```
data(bc_longlat_map_img_ras)
```

`cdlm_robot` *Shiny app for 1D simulation of robot movement with CDLM.*

Description

Shiny app for 1D simulation of robot movement with CDLM.

Usage

```
cdlm_robot()
```

Details

This shiny app illustrates a 1-D robot movement model. Here, T ("maximum number of iterations") steps are simulated for a robot moving along a 1-D line. Each of the T steps represents a length of time represented by "time step (sec)" seconds; the longer the interval, the more location uncertainty there is between steps. At each step, the robot moves with velocity (v_t) modeled by a normal distribution with mean alpha ("unknown true mean of velocity") and variance "known true variance." For simplicity, we will only attempt to model the mean velocity while sequentially observing only the locations, since we assume the variance is known. The particle filter learns the movement parameters through N ("number of particles") particles, or independent simulations. At each point in time, the filter simulates N draws of the velocity from the prior distribution, a normal distribution with mean mu ("prior mean on velocity mean") and variance sigma ("prior variance on velocity mean"). These distributions are shown in color panel 1. Ideally, over time the colored distributions should converge to the true one (thick black curve).

Panel 1 shows the particles' distributions of the velocities. Ideally the means of these distributions should converge to the true value (vertical line). Note: this simulation works best if the distribution of true velocity is either clearly negative or positive. If the distribution straddles v_t=0 with significant probability, movement will be more difficult to visualize. In the above case, the location should be the one more in the direction of the sign of velocity (i.e. if velocity distribution >0, then the robot should be consistently moving to the right).

Panel 2 shows each particle's prediction of the location (black dot) and the 95% confidence interval of this prediction (width of colored rectangle). The true observed location and the previous one are the two dashed vertical lines. Particles whose dots fall closer to the vertical line have better prediction.

Panel 3 shows the weights of the particles as calculated by the closeness of their location prediction to the true one. Closer particles in panel 2 should have higher weights (the colors correspond).

Panel 4 shows the predictions and confidence intervals of particles being resampled by their weights (with replacement). More of the predictions should be closer to the truth here than in panel 2.

Panel 5 shows convergence over time of the means of the particle distributions of velocity (panel 1) to the true value (vertical dashed line). Ideally these should converge to the true value.

Panel 6 shows the history of predicted locations over time, by vertical lines representing each particle's predictions. Ideally the particle predictions should both converge to the observed locations and should also be grouped closer together, as the estimated velocity distribution standard deviation decreases.

The particle filter models the true value of location and the true velocity. It is reasonable to assume that these are independent, which is why the covariance matrices given are diagonal.

Note

Video explanation of simulation applet by author: `https://youtu.be/iVG_bCU0jCA`

References

Ackerman, Samuel. "A Probabilistic Characterization of Shark Movement Using Location Tracking Data." Temple University doctoral thesis, 2018. `https://digital.library.temple.edu/digital/collection/p245801coll10/id/499150`

Carvalho, Carlos M., Johannes, Michael S., Lopes, Hedibert F., and Nicholas G. Polson. "Particle learning and smoothing." Statistical Science, 2010.

`cdlm_robot_twostate` *Shiny app for simulation of 1D robot movement with CDLM and two states.*

Description

Shiny app for simulation of 1D robot movement with CDLM and two states.

Usage

```
cdlm_robot_twostate()
```

Details

See `cdlm_robot` for explanation of the basic concepts. This function is similar except there are two behavioral states (1 and 2, "slow"/"fast") to model, as well as the switching probabilities between them.

The means of the velocities of the two behaviors are simulated by a normal distribution with two means alpha ("unknown true mean of velocity", types 1 and 2). The variance in each case is the same and known, as before. The prior means and variances of the velocities are assigned as before.

The transition probabilities between the behaviors are given by "transition probability between type 1 and 2" and "2 and 1". If box "are transition probabilities known?" is checked, then they are known. Otherwise, the transition probabilities will be estimated by a Dirichlet prior (vector "Dirichlet prior values" of form 1->1, 1->2, 2->1, 2->2 of positive numbers, which should roughly correspond to the true probabilities in ratio). Note that the predictions in this simulation are unlikely to be as good as in the prior 1-D example since there are more parameters to learn and only a limited number of timesteps or particles.

Panel 1 shows the particles' distributions of the velocities for each behavior. This simulation works best if the distributions are well-separated.

Panel 2 shows the location predictions for each behavior. The black dot indicates the mean predicted location, and the rectangle width is the width of the 95% confidence interval. The rectangle for behavior 1 is solid, for behavior 2 it has crosshatches.

Panel 3 shows the overall resampling weights for the particles, as well as the behavior-conditional ones. The higher the behavior-conditional weight bar is, the better the particle's prediction at that behavior matches what was observed. The overall weight (top row) is the average of the conditional weight values, weighted by the transition probability into that behavior.

Panel 4 shows the resampled particles, along with their prediction of location and behavior type. Ideally, the resampled rectangles should be centered around the observed point. It is not necessarily true that the resampled (most likely) rectangles will be the narrowest, since the likelihood of the behavior predicting the observed location is a combination of both the density of that location at the prediction distribution (closeness to the center), as well as the likelihood (transition probability) of having that behavior, given the previous one. In panel 4, the particle predictions are shown one at a time as that particle is resampled; the weight bar in panel 3 should be in bold as that particle is selected.

Panel 5 shows convergence over time of the means of the particle distributions of velocity (panel 1) to the true value (vertical dashed line). Ideally these should converge to the true value.

Panel 6 shows the history of predicted locations over time, by vertical lines representing each particle's predictions. Ideally the particle predictions should both converge to the observed locations and should also be grouped closer together, as the estimated velocity distribution standard deviation decreases.

Panel 7 shows the estimated distributions of the behavior switching probabilities (if they are not known). The true probabilities are shown by a vertical line, and ideally the mean of the estimated distribution should be around there.

Panel 8 shows the accuracy of particle predictions of the behavior. The color (1=black, 2=gray) is the true behavior type, and the height of the bar is the fraction of particles correctly predicting it. Ideally, all bars should be high.

Note

Video explanation of simulation applet by author: `https://youtu.be/4XR8eB89z7E`

References

Ackerman, Samuel. "A Probabilistic Characterization of Shark Movement Using Location Tracking Data." Temple University doctoral thesis, 2018. `https://digital.library.temple.edu/digital/collection/p245801coll10/id/499150`

Carvalho, Carlos M., Johannes, Michael S., Lopes, Hedibert F., and Nicholas G. Polson. "Particle learning and smoothing." Statistical Science, 2010.

`cdlm_robot_twostate_2D` *Shiny app for simulation of 2D robot movement with CDLM and two states.*

Description

Shiny app for simulation of 2D robot movement with CDLM and two states.

Usage

```
cdlm_robot_twostate_2D()
```

Details

See `cdlm_robot` and `cdlm_robot_twostate` for explanation of the basic concepts. This function simulates a 2-D moving robot with two behavioral states (1 and 2, "slow"/"fast") to model, as well as the switching probabilities between them.

The means of the log-speeds of the two behaviors are simulated by a normal distribution with two means alpha ("unknown true mean of log-speed", types 1 and 2). The variance in each case is the same and known, as before. The prior means and variances of the velocities are assigned as before.

The transition probabilities between the behaviors are given by "transition probability between type 1 and 2" and "2 and 1". If box "are transition probabilities known?" is checked, then they are known. Otherwise, the transition probabilities will be estimated by a Dirichlet prior (vector "Dirichlet prior values" of form 1->1, 1->2, 2->1, 2->2 of positive numbers, which should roughly correspond to the true probabilities in ratio). Note that the predictions in this simulation are unlikely to be as good as in the prior 1-D example since there are more parameters to learn and only a limited number of timesteps or particles.

Panel 1 shows the particles' distributions of the log-speed for each behavior. This simulation works best if the distributions are well-separated.

Panel 2 shows the location predictions (center with confidence ellipse, either solid or dashed by behavior type)

Panel 3 shows the overall resampling weights for the particles, as well as the behavior-conditional ones. The higher the behavior-conditional weight bar is, the better the particle's prediction at that behavior matches what was observed. The overall weight (top row) is the average of the conditional weight values, weighted by the transition probability into that behavior.

Panel 4 shows the resampled particles, along with their prediction of location and behavior type. Ideally, the resampled ellipses should be centered around the observed point. It is not necessarily true that the resampled (most likely) ellipses will be the smallest, since the likelihood of the behavior predicting the observed location is a combination of both the density of that location at the ellipse, as well as the likelihood (transition probability) of having that behavior, given the previous one. In panel 4, the particle predictions are shown one at a time as that particle is resampled; the weight bar in panel 3 should be in bold as that particle is selected.

Panel 5 shows convergence over time of the means of the particle distributions of log-speed (panel 1) to the true value (vertical dashed line). Ideally these should converge to the true value.

Panel 6 shows the history of predicted locations over time in terms of a spatial density plot (grayscale shading). Ideally, these should concentrate around the red overlaid trajectory of observed locations.

Panel 7 shows the estimated distributions of the behavior switching probabilities (if they are not known). The true probabilities are shown by a vertical line, and ideally the mean of the estimated distribution should be around there.

Panel 8 shows the accuracy of particle predictions of the behavior. The color (1=black, 2=gray) is the true behavior type, and the height of the bar is the fraction of particles correctly predicting it. Ideally, all bars should be high.

Note

Video explanation of simulation applet by author: https://youtu.be/4XR8eB89z7E

References

Ackerman, Samuel. "A Probabilistic Characterization of Shark Movement Using Location Tracking Data." Temple University doctoral thesis, 2018. https://digital.library.temple.edu/digital/collection/p245801coll10/id/499150

Carvalho, Carlos M., Johannes, Michael S., Lopes, Hedibert F., and Nicholas G. Polson. "Particle learning and smoothing." Statistical Science, 2010.

`EKF_1d_interp_joint` *Extended Kalman Filter (EKF) for 1-D movement with interpolation*

Description

Extended Kalman Filter (EKF) for 1-D movement with interpolation

Usage

```
EKF_1d_interp_joint(d, npart=100, sigma_pars,
                    alpha0_pars=list(mu0=c(5, 9), V0=c(0.25, 0.25)),
                    Errvar0=rep(list(5), 2), Errvar_df=c(20, 20),
                    Particle_errvar0, Particle_err_df=20, delaysample=1,
                    dirichlet_init=c(10,3,3,8), maxStep=NULL,
                    state_favor=c(1,1), nstates=2,
                    lowvarsample=FALSE, time_radius=60*30, spat_radius=300,
                    min_num_neibs=10, interact=TRUE,
                    interact_pars=list(mu0=0, precision0=2,
                    known_precision=2),
                    neff_sample=1, time_dep_trans=FALSE,
                    time_dep_trans_init=dirichlet_init, smoothing=FALSE,
                    fix_smoothed_behaviors=TRUE, smooth_parameters=TRUE,
                    reg_dt=120, max_int_wo_obs=NULL,
                    resamp_full_hist=TRUE, compare_with_known=FALSE,
                    known_trans_prob=NULL, known_foraging_prob=NULL,
                    known_regular_step_ds=NULL, update_eachstep=FALSE,
                    update_params_for_obs_only=FALSE,
                    output_plot=TRUE, loc_pred_plot_conf=0.5,
                    output_dir=getwd(), pdf_prefix="EKF_1D", verbose=3)
```

Arguments

`d`	Dataset of observations, with required variable columns: tag, X, velocity, date_as_sec, time_to_next, state.guess2, prev.guess2.
`npart`	Number of particles to be used in simulation.
`sigma_pars`	Vector of inverse-gamma parameters for sigma^2 (logV variance). Two elements for each state. The inverse gamma parameters are specified in pairs.
`alpha0_pars`	List of initial values of mean velocity (mu) and degrees of freedom (V), one for each behavioral state.
`Errvar0`	List of prior 1x1 covariance matrices for predicting y from x, one for each behavioral state.
`Errvar_df`	Vector of degrees of freedom of `Errvar0` covariance matrices.
`Particle_errvar0`	Prior 1x1 covariance matrix for predicting x_t from x_t-1.
`Particle_err_df`	Degree of freedom of `Particle_errvar0`.
`dirichlet_init`	List of 4-element vectors specifying Dirichlet parameters for transition matrices for each region. Will be replicated to equal number of regions.
`maxStep`	Maximum number of regular steps to simulate. Default is NULL, meaning that the number of regular steps simulated will be the minimum number required to cover the range of observed data. If not NULL, maxStep will be the minimum of the submitted value or the the above.
`delaysample`	Number of regular steps at which resampling will begin. The default =1 means resampling will begin immediately.
`state_favor`	Vector of weights to favor states when resampling (but not propagating). For instance c(1,3) will favor state 2 weight 3 times as much as state 1 weights for particles. By default, they are equally weighted.
`nstates`	Number of behavioral states. For now restricted to a maximum of 2.
`lowvarsample`	Logical. If TRUE, use low-variance sampling when resampling particles to ensure particles are resampled proportionately to weight. Otherwise there is some sampling variance when drawing random samples. The setting applies to smoothing as well.
`time_radius`	Time in seconds to consider for spatial neighbors (1-D interval on either side).
`spat_radius`	Radius (half of interval length) in meters of spatial neighborhood.
`min_num_neibs`	Minimum number of time and spatial radius observations that need to exist to constitute a neighborhood.
`interact`	Logical. If TRUE, simulate interaction parameters of neighborhood. If `nstates=1`, automatically set to FALSE.
`interact_pars`	List of interaction priors: `mu0` and `precision0` are prior mu and precision for normal draws of interaction parameter. `known_precision` is the known precision of the lognormal intensity.
`neff_sample`	Number between 0 and 1. If effective sample size < `neff_sample`, then resample. Recommended to always resample if interpolating, so set neff_sample=1 as default.

`time_dep_trans`	Logical. If TRUE, state transition matrices are time-dependent meaning that probability depends on the number of steps a shark has remained in the current state.
`time_dep_trans_init`	4-element numeric vector of Dirichlet parameters for `time_dep_trans`.
`smoothing`	Logical. If TRUE, perform smoothing at the end.
`fix_smoothed_behaviors`	Logical. If TRUE, when performing smoothing, keep behavior modes fixed for each particle history from what was originally predicted duruing filtering, before smoothing. This means the particles will be smoothed backwards with each particle weight at each time point being conditioned on the behavior predicted in filtering. Thus, the behavioral agreement with, say, the observed or true behaviors is the same for smoothing as for filtering, since behaviors are not allowed to change. If `nstates==1`, then automatically `fix_smoothed_behaviors=FALSE`.
`smooth_parameters`	Logical. If TRUE, when performing smoothing, resample the parameters theta as well.
`reg_dt`	Length in seconds of each regular interval.
`max_int_wo_obs`	When simulating, the maximum number of intervals of length `reg_dt` without observations for a given shark that we will simulate. If this is exceeded, algorithm will wait until next observation and start from there. Default is NULL, meaning it will be set to `maxStep`, and thus the algorithm will continue simulating without stopping, regardless of when the next observation is.
`resamp_full_hist`	Logical. If TRUE, resample the full particle history, not just all particle times since the last observation, each time resampling occurs.
`compare_with_known`	Logical. If TRUE, provide a known regular-step dataset from which `d` is a irregularly-sampled subset, for comparison with particle predictions.
`known_trans_prob`	If `nstates`=2, a matrix of row 2 where each column is the behavior transition probabilities between each opposing behavior, in each region.
`known_foraging_prob`	If `nstates`=2, a matrix of the foraging probabilities for each region.
`known_regular_step_ds`	If `compare_with_known`=TRUE, the dataset of the original regular-step trajectories. Note: this dataset needs to have column `date_as_sec` (date in seconds) and time gap `reg_dt` be the same as the set of regular-step intervals that the EKF is trying to estimate movement at. Otherwise, the simulated movement locations and the true ones will not correspond.
`update_eachstep`	Logical. If TRUE, for regular steps without observations, update the movement parameters based on the simulated movements. If FALSE, parameters are only updated based on the simulated movements when a new observation occurs; this means the simulated movements are drawn using the parameter values learned since the last observation.

`update_params_for_obs_only`	Logical. If TRUE, the particle movement parameters are updated based on simulated movement only at intervals with observed locations. If FALSE, particle movement in intermediate steps that are simulated will be used to update as well. If TRUE, then `update_eachstep=FALSE`, meaning that parameter updates will be done only for (and at) the steps that represent observations. If FALSE, then `update_eachstep` can be either TRUE or FALSE, but simulated steps will be used to update, but `update_eachstep` controls the timing of the the update; if TRUE, it happens one step at a time, and if FALSE, a batch update is done at observations for the set of simulated steps.
`output_plot`	Logical. If TRUE, a set of diagnostic plots will be printed to a file in `output_dir`. Otherwise, it will be output to the plotting console.
`loc_pred_plot_conf`	Numeric. Confidence level of confidence interval for location prediction error to plot in step-wise diagnostics.
`pdf_prefix`	String prefix for output PDF filename, if `output_plot = TRUE`. Filename will be the prefix followed by a timestamp.
`output_dir`	Directory for output PDF of diagnostic plots.
`verbose`	Integer, one of 0,1,2,3. Control of verbosity of printouts during simulation. 3 means show both printouts and plots; 2 means show plots only; 1 means show printouts only; 0 means show no plots or prinouts. Final plotting will be done regardless.

Value

`d`	Input dataset as `data.frame`
`N`	Number of regular steps of length `reg_dt` needed to cover the observed range of time.
`t_reg`	Vector of times of regular step `reg_dt`.
`nsharks`	Number of sharks in output data.
`shark_names`	Names of sharks in output data.
`shark_valid_steps`	List of regular-step intervals that each shark has simulated particle movement for.
`shark_intervals`	List of regular-step intervals that each shark has observations for.
`first_intervals`	List of regular-step intervals that begin each shark's segments of simulated particle movement. If observed gaps are larger than `max_int_wo_obs`, the shark's trajectory will be simulated as two or more separate segments.
`included_intervals`	Unique list of regular-step intervals with simulated movement for any shark.
`mu`	Array of estimated values of mean log-velocity for normal inverse-gamma conjugate distribution

`XY_errvar`	Estimated matrix and degrees of freedom of estimated location error covariance, for each behavior.
`sigma_pars`	Posterior inverse gamma distribution parameters for the velocity (or, for 2-D, log-velocity) variance.
`Xpart_history`	Overall history of estimated movement values.
`param_draws`	Posterior sampled values of mean of velocity (or, for 2-D, log-velocity).
`variance_draws`	Posterior sampled value of variance of velocity (or, for 2-D, log-velocity).
`eff_size_hist`	History of effective sample sizes in simulations.
`agree_table`	Table of observed agreement between particle predictions of behavior and those observed, overall and by behavior, if `nstates` > `1`.
`states`	Observed vector of behavioral states.
`state_counts`	Array of total number of simulated regular-step intervals in each behavioral state.
`lambda_matrix`	History of particle predicted values of lambda, the behavior variable.
`lambda_matrix_beforesamp`	Same as `lambda_matrix`, except the history before each time has not been resampled according to the particle resampling weights. For `lambda_matrix`, `Xpart_history`, and other estimated outputs, the entire particle history is resampled.
`resample_history`	Fraction of unique particles that are resampled at each regular step over the history.
`transition_mat`	Estimated transition probability matrix parameters for Dirichlet distribution. If `nstates==1`, is meaningless.
`error_beforesamp`	For each regular step `i` with an observation, the quantiles of summed prediction errors before each round of resampling, across history.
`error_beforesamp_quantiles`	Quantiles of `error_beforesamp_allpart` across history.
`error_final_allpart`	For each regular step `i` with an observation, the sum of prediction errors for any observations in that interval (final after resampling).
`error_final_quantiles`	Quantiles of `error_final_allpart` across history.
`error_true_allpart`	If `compare_with_known` == `TRUE`, for each regular step `i`, the sum of prediction errors for any true locations in that interval.
`error_true_quantiles`	If `compare_with_known` == `TRUE`, quantiles of `error_final_true_allpart` across history.

The following inputted parameters are returned :

`npart`

nstates

state_favor

known_regular_step_ds

known_foraging_prob

neff_sample

resamp_full_hist

time_dep_trans

interact

spat_radius

time_radius

lowvarsample

update_eachstep

update_params_for_obs_only

The following are returned if `nstates` > 1:

`trans_counts`	Array of total number of simulated regular-step intervals with transitions between each possible pair of behaviors.
`trans_mean`	Posterior estimates of mean behavior switching probabilities from `region_trans_draws`.
`region_foraging_draws`	Posterior estimate of probability of foraging (lambda=0) from behavior switching probabilities.
`region_trans_draws`	Posterior draws of behavior switching probabilities from `transition_mat`. For 2-D, this is separately by region, if there are multiple regions.

In addition, the following are returned if `compare_with_known` = `TRUE`:

`error_final_true_allpart`	Errors from estimating true locations from particle locations (at the same times).
`error_final_true_quantiles`	Quantiles of `error_final_true_allpart` across history.
`euclidean_estimate_true_from_obs`	Estimates of true locations by Euclidean interpolation from observations
`error_euclidean_estimate_true_from_obs`	Euclidean error from `euclidean_estimate_true_from_obs` compared to true locations from `known_regular_step_ds`.

In addition, the following are returned if `interact` = `TRUE`:

`spatial_interact_pars`	Estimated parameters for sharks' tendency to be influenced by other neighboring sharks in determining behavior.

`interact_mu_draws`
: Posterior sampled values of interaction mu parameter.

`interact_intensity_draw`
: Posterior sampled values of interaction tendency multiplier, at different proportions of neighboring sharks with second behavior type.

`spatial_interact_mu_history`
: History of simulated values of interaction mu.

`spatial_interact_intensity_history`
: History of simulated values of interaction tendency multiplier.

The following are returned if `smoothing = TRUE`:

`Xpart_history_smoothed`
: Resampled values of `Xpart_history` by reverse smoothing resampling (see Carvalho et al).

`error_smoothed_allpart`
: For each regular step `i`, the sum of prediction errors for smoothed particles for any observations in that interval.

`error_smoothed_quantiles`
: Quantiles of `error_smoothed_allpart` across history.

In addition, if `smooth_parameters = TRUE`:

`param_draws_smoothed`
: Posterior sampled values of mean of velocity (or, for 2-D, log-velocity) after resampling by smoothing.

`variance_draws_smoothed`
: Posterior sampled values of variance of velocity (or, for 2-D, log-velocity) after resampling by smoothing.

`transition_mat_smoothed`
: Estimated transition probability matrix parameters for Dirichlet distribution after resampling by smoothing.

In addition, if `smooth_parameters = TRUE` and `interact = TRUE`:

`spatial_interact_pars_smoothed`
: Estimated parameters for sharks' tendency to be influenced by other neighboring sharks in determining behavior, after resampling by smoothing.

`interact_mu_draws_smoothed`
: Posterior sampled values of interaction mu parameter, after resampling by smoothing.

`interact_intensity_draw_smoothed`
: Posterior sampled values of interaction tendency multiplier, at different proportions of neighboring sharks with second behavior type, after resampling by smoothing.

In addition to smoothing, if `compare_with_known = TRUE`:

`error_smoothed_true_allpart`
: For each regular step `i`, the sum of prediction errors for smoothed particles for any observations in that interval.

`error_smoothed_true_quantiles`	Quantiles of `error_smoothed_true_allpart` across history.

In addition to smoothing, if `smoothing = TRUE` but `fix_smoothed_behaviors = FALSE` (smoothed behaviors allowed to change from filtering):

`mu_smoothed`	Corresponding version of `mu` after resampling by smoothing.
`sigma_pars_smoothed`	Corresponding version of `sigma_pars` after resampling by smoothing.
`agree_table_smoothed`	Corresponding version of `agree_table` for smoothed states `lambda_matrix_smoothed`.

Note

See `sim_trajectory_joint` for a full example of usage. Video explanation of EKF state-space model by author: `https://youtu.be/SgyhRVUn77k`

Author(s)

Samuel Ackerman

References

Ackerman, Samuel. "A Probabilistic Characterization of Shark Movement Using Location Tracking Data." Temple University doctoral thesis, 2018. `https://digital.library.temple.edu/digital/collection/p245801coll10/id/499150`

Carvalho, Carlos M., Johannes, Michael S., Lopes, Hedibert F., and Nicholas G. Polson. "Particle learning and smoothing." Statistical Science, 2010.

`EKF_interp_joint` *Extended Kalman Filter (EKF) for joint shark movement with interpolation*

Description

Extended Kalman Filter (EKF) for joint shark movement with interpolation

Usage

```
EKF_interp_joint(area_map=NULL, d, npart=100, sigma_pars, tau_pars,
                 mu0_pars=list(alpha=c(-4.5 ,-2), beta=c(0,0)),
                 V0_pars=list(alpha=c(0.25, 0.25), beta=c(0.25, 0.25)),
                 Errvar0=rep(list(diag(2)), 2),
                 Errvar_df=c(20, 20), Particle_errvar0, Particle_err_df=20,
                 dirichlet_init=c(9,2,2,7), logvelocity_truncate=c(-10, 15),
                 maxStep=NULL, delaysample=1, state_favor=c(1,1),
                 nstates=2,centroids=matrix(c(0,0), ncol=2),
                 truncate_to_map=TRUE, enforce_full_line_in_map=TRUE,
                 do_trunc_adjust=TRUE, lowvarsample=TRUE,
                 time_radius=60*30, spat_radius=300, min_num_neibs=10,
                 interact=TRUE, interact_pars=list(mu0=0, precision0=2,
                 known_precision=2), neff_sample=1, time_dep_trans=FALSE,
                 time_dep_trans_init=dirichlet_init, smoothing=FALSE,
                 fix_smoothed_behaviors=TRUE, smooth_parameters=TRUE,
                 reg_dt=120, max_int_wo_obs=NULL, resamp_full_hist=TRUE,
                 compare_with_known=FALSE, known_trans_prob=NULL,
                 known_foraging_prob=NULL, known_regular_step_ds=NULL,
                 update_eachstep=FALSE, update_params_for_obs_only=FALSE,
                 output_plot=TRUE, loc_pred_plot_conf=0.5,
                 output_dir=getwd(), pdf_prefix="EKF_2D", verbose=3)
```

Arguments

`area_map`	Shapefile within which the observations are located (optional). Should be the output of applying `sf::st_geometry` on an object of class `sf`. If input is NULL, a default rectangular one is created that contains the observed X-Y points in `d`.
`d`	Dataset of observations, with required variable columns: tag, X, Y, logvelocity, speed, turn.angle.rad, region (optional), date_as_sec, time_to_next, state.guess2, prev.guess2.
`npart`	Number of particles to be used in simulation.
`sigma_pars`	Vector of inverse-gamma parameters for sigma^2 (logV variance). Two elements for each state. The inverse gamma parameters are specified in pairs.
`tau_pars`	Vector of inverse-gamma parameters for tau^2 (turn angle variance).
`mu0_pars`	List of initial values of mean logV (alpha) and turn (beta) for one or two behavioral states.
`V0_pars`	List of initial values of degrees of freedom of inverse-gamma sigma and tau (variances of alpha and beta) for one or two behavioral state.
`Errvar0`	List of prior 2x2 covariance matrices for predicting y from x, one for each behavioral state.
`Errvar_df`	Vector of degrees of freedom of `Errvar0` covariance matrices.
`Particle_errvar0`	Prior 2x2 covariance matrix for predicting x_t from x_t-1.
`Particle_err_df`	Degree of freedom of `Particle_errvar0`.
`dirichlet_init`	List of 4-element vectors specifying Dirichlet parameters for transition matrices for each region. Will be replicated to equal number of regions.
`logvelocity_truncate`	When simulating log-velocity, a vector of the allowable range (values outside will be truncated to fall in this range). Log-velocity is simulated by a normal distribution (which is symmetric but can be positive or negative), so that speed (=exp(log_velocity)) will be positive. However, the transformation has asymmetric impact in that, say, a fixed error in underestimating log-velocity results in a smaller displacement (when translated to speed and thus distance) than the

	same error over-estimated. The variance of log-velocity takes into account low and high values equally. This restriction prevents the variance from growing too large from low (e.g. very negative) values of log-velocity, which will then cause large over-estimates of speed and distance traveled. The difference between, say, log-velocity of -2 and -50 is very small in practical terms of distance, but the effect on the variance will be much larger for the -50.
`maxStep`	Maximum number of regular steps to simulate. Default is NULL, meaning that the number of regular steps simulated will be the minimum number required to cover the range of observed data. If not NULL, maxStep will be the minimum of the submitted value or the the above.
`delaysample`	Number of regular steps at which resampling will begin. The default =1 means resampling will begin immediately.
`state_favor`	Vector of weights to favor states when resampling (but not propagating). For instance c(1,3) will favor state 2 weight 3 times as much as state 1 weights for particles. By default, they are equally weighted.
`nstates`	Number of behavioral states. For now restricted to a maximum of 2.
`centroids`	Matrix with two columns specifying the centroids of regions.
`truncate_to_map`	Logical. If TRUE, make sure that coordinate predictions are inside the boundary `area_map` by truncated sampling.
`enforce_full_line_in_map`	Logical. If TRUE, when conducting truncated sampling (`truncate_to_map==TRUE`), count the prediction of the next location as being inside the boundary if the full line segment connecting it to the current location is inside the map. Otherwise, only the predicted point (and not the line connecting them) must be inside the map. The idea is that the truncation allows only 'feasible' straight-line moves to be made, and so the full line segment must be inside the map. However, there may be situations in which this restriction prevents the algorithm from making good predictions, such as if the time gap `reg_dt` is too long, or if the map contains 'narrow' areas where requiring the line to be inside would prevent a prediction and the algorithm would get 'stuck'.
`do_trunc_adjust`	Logical. If TRUE, adjust particle posterior weights by the fraction of their predictions that are within the truncation boundary.
`lowvarsample`	Logical. If TRUE, use low-variance sampling when resampling particles to ensure particles are resampled proportionately to weight. Otherwise there is some sampling variance when drawing random samples. The setting applies to smoothing as well.
`time_radius`	Time in seconds to consider for spatial neighbors.
`spat_radius`	Radius in meters of (circular) spatial neighborhood.
`min_num_neibs`	Minimum number of time and spatial radius observations that need to exist to constitute a neighborhood.
`interact`	Logical. If TRUE, simulate interaction parameters of neighborhood. If `nstates=1`, or if only one shark, automatically set to FALSE.

`interact_pars`	List of interaction priors: `mu0` and `precision0` are prior mu and precision for normal draws of interaction parameter. `known_precision` is the known precision of the lognormal intensity.
`neff_sample`	Number between 0 and 1. If effective sample size < `neff_sample`, then resample. Recommended to always resample if interpolating, so set neff_sample=1 as default.
`time_dep_trans`	Logical. If TRUE, state transition matrices are time-dependent meaning that probability depends on the number of steps a shark has remained in the current state.
`time_dep_trans_init`	4-element numeric vector of Dirichlet parameters for `time_dep_trans`.
`smoothing`	Logical. If TRUE, perform smoothing at the end.
`fix_smoothed_behaviors`	Logical. If TRUE, when performing smoothing, keep behavior modes fixed for each particle history from what was originally predicted during filtering, before smoothing. This means the particles will be smoothed backwards with each particle weight at each time point being conditioned on the behavior predicted in filtering. Thus, the behavioral agreement with, say, the observed or true behaviors is the same for smoothing as for filtering, since behaviors are not allowed to change. If `nstates==1`, then automatically `fix_smoothed_behaviors=FALSE`.
`smooth_parameters`	Logical. If TRUE, when performing smoothing, resample the parameters theta as well.
`reg_dt`	Length in seconds of each regular interval.
`max_int_wo_obs`	When simulating, the maximum number of intervals of length `reg_dt` without observations for a given shark that we will simulate. If this is exceeded, algorithm will wait until next observation and start from there. Default is NULL, meaning it will be set to `maxStep`, and thus the algorithm will continue simulating without stopping, regardless of when the next observation is.
`resamp_full_hist`	Logical. If TRUE, resample the full particle history, not just all particle times since the last observation, each time resampling occurs.
`compare_with_known`	Logical. If TRUE, provide a known regular-step dataset from which `d` is a irregularly-sampled subset, for comparison with particle predictions.
`known_trans_prob`	If `nstates = 2`, a matrix of row 2 where each column is the behavior transition probabilities between each opposing behavior, in each region.
`known_foraging_prob`	If `nstates = 2`, a matrix of the foraging probabilities for each region.
`known_regular_step_ds`	If `compare_with_known = TRUE`, the dataset of the original regular-step trajectories. Note: this dataset needs to have column `date_as_sec` (date in seconds) and time gap `reg_dt` be the same as the set of regular-step intervals that the EKF is trying to estimate movement at. Otherwise, the simulated movement locations and the true ones will not correspond.

`update_eachstep`
: Logical. If TRUE, for regular steps without observations, update the movement parameters based on the simulated movements. If FALSE, parameters are only updated based on the simulated movements when a new observation occurs; this means the simulated movements are drawn using the parameter values learned since the last observation.

`update_params_for_obs_only`
: Logical. If TRUE, the particle movement parameters are updated based on simulated movement only at intervals with observed locations. If FALSE, particle movement in intermediate steps that are simulated will be used to update as well. If TRUE, then `update_eachstep = FALSE`, meaning that parameter updates will be done only for (and at) the steps that represent observations. If FALSE, then `update_eachstep` can be either TRUE or FALSE, but simulated steps will be used to update, but `update_eachstep` controls the timing of the the update; if TRUE, it happens one step at a time, and if FALSE, a batch update is done at observations for the set of simulated steps.

`output_plot`
: Logical. If TRUE, a set of diagnostic plots will be printed to a file in `output_dir`. Otherwise, it will be output to the plotting console.

`loc_pred_plot_conf`
: Numeric. Confidence level of ellipse for location prediction error to plot in step-wise diagnostics.

`pdf_prefix`
: String prefix for output PDF filename, if `output_plot = TRUE`. Filename will be the prefix followed by a timestamp.

`output_dir`
: Directory for output PDF of diagnostic plots.

`verbose`
: Integer, one of 0,1,2,3. Control of verbosity of printouts during simulation. 3 means show both printouts and plots; 2 means show plots only; 1 means show printouts only; 0 means show no plots or prinouts. Final plotting will be done regardless.

Value

Many of the returned values are the same as in `EKF_1d_interp_joint`. The ones that differ are listed below.

`centroids`
: Input centroids of spatial regions.

`nregions`
: Number of unique regions, as determined by `centroids`

.

`tau_pars`
: Posterior inverse gamma distribution parameters for the turn angle variance.

`cov_err_hist`
: Overall history of location estimate error draws.

`param_draws`
: Posterior sampled valued of mean of log-velocity and turn.

`variance_draws`
: Posterior sampled valued of variance of log-velocity and turn.

`trans_mean_byregion`
: Posterior estimates of mean behavior switching probabilities from `region_trans_draws`.

`region_counts`	Array of total number of simulated regular-step intervals that shark begin movement in each spatial region. A proxy for the total amount of time spent in each region.
`euclidean_estimate_true_from_obs`	Estimates of true locations by Euclidean and Bezier cubic spline interpolation from observations
`error_euclidean_estimate_true_from_obs`	Euclidean error from `euclidean_estimate_true_from_obs` compared to true locations from `known_regular_step_ds`.

The following inputted parameters are returned:

`area_map`

Note

See `sim_trajectory_joint` for a full example of usage. Video explanation of EKF state-space model by author: `https://youtu.be/SgyhRVUn77k`

Author(s)

Samuel Ackerman

References

Ackerman, Samuel. "A Probabilistic Characterization of Shark Movement Using Location Tracking Data." Temple University doctoral thesis, 2018. `https://digital.library.temple.edu/digital/collection/p245801coll10/id/499150`

Carvalho, Carlos M., Johannes, Michael S., Lopes, Hedibert F., and Nicholas G. Polson. "Particle learning and smoothing." Statistical Science, 2010.

`low_var_sample` *Sample particles using low-variance sampling.*

Description

Sample particles using low-variance sampling.

Usage

```
low_var_sample(wts, M=length(wts))
```

Arguments

`wts`	Vector of weights.
`M`	The number of items to sample. When resampling, should be number of particles `npart`.

Details

Low-variance sampling guarantees items will be sampled in proportion to their weights. With random sampling with replacement (`sample` function), there is some variability in the final proportions of items.

Value

A numeric vector of length `M`.

Author(s)

Samuel Ackerman

References

James Edward Baker. Reducing bias and inefficiency in the selection algorithm. Proceedings of the Second International Conference on Genetic Algorithms on Genetic Algorithms and Their Application, 1987.

Examples

```
n <- 20
w <- runif(n)

#can use M != to length(wts)
low_var_sample(wts=w, M=15)

lv <- low_var_sample(wts=w, M=n)

#usual resampling with weights
a <- sample(x=1:n, size=n, prob=w, replace=TRUE)

#the first should be more in proportion to weights
table(lv)/n
table(a)/n
#proportions
w/sum(w)
```

`make_segments` *Plot path connecting points on ggplot.*

Description

Convert x-y coordinates to a `data.frame` for `geom_path` plotting on `ggplot`.

Usage

```
make_segments(xy, N=nrow(xy))
```

Arguments

`xy`	Matrix or `data.frame` of x-y points to plot as a path. If there are any rows that are NA-valued, the resulting path will consist of disconnected segments in those locations.
`N`	Number of rows of `xy` to plot as a path. By default, the number of rows, so all of `xy`.

Author(s)

Samuel Ackerman

Examples

```
#generate toy dataset
library(ggplot2)
d <- as.data.frame(cbind(X=runif(50), Y=runif(50)))

#create segments with some missing lines so will be some gaps)
dseg <- d
dseg[ sample(20),] <- NA

g <- ggplot(d, aes(x=.data$X, y=.data$Y)) + theme_bw()
g <- g + stat_density2d(aes(fill=after_stat(!!str2lang("density"))), geom="tile", contour=FALSE)
g <- g + scale_fill_gradient(low="white", high="black") + theme(legend.position="bottom")
g <- g + geom_path(data=make_segments(xy=dseg), aes(x=.data$X, y=.data$Y), colour="red", lwd=1.5)
g
```

`normalize_angle` *Wrap angle measurements to the interval (-pi, pi).*

Description

Wrap angle measurements to the interval (-pi, pi).

Usage

```
normalize_angle(theta)
```

Arguments

`theta`	Numeric vector.

Author(s)

Samuel Ackerman

Examples

```
x <- rnorm(n=1000, mean=1, sd=2)
xn <- normalize_angle(x)

plot(density(x), xlab="x", main="Unwrapped and wrapped normal density", las=1)
abline(v=1)

#this density is only estimated from -pi to pi
dens_wrapped <- density(xn, from=-pi, to=pi)
lines(dens_wrapped, col="red")

segments(x0=c(-pi, pi), x1=c(-pi, pi), y0=c(0,0),
y1=dens_wrapped$y[c(1, length(dens_wrapped$y))],
col="red")

legend("topleft", col=c(1,2), legend=c("unwrapped","wrapped"), lty=1)
```

`rug_multicolor` *Multicolor rug of tick marks.*

Description

Adapt `rug` function to allow tick marks to be of different colors.

Usage

```
rug_multicolor(x, plot_side=3, ticksize=-0.04, col_vec=rep(1, length(x)))
```

Arguments

`x`	Numeric vector of axis tick mark locations.
`plot_side`	Which side to plot on. 1=bottom, 2=left, 3=top, and 4=right.
`ticksize`	Size of tick marks. Negative values mean ticks are on outside of plot. This feeds into the `tck` parameter of the `axis` function.
`col_vec`	Vector of color definitions, corresponding to each value of `x`.

Author(s)

Samuel Ackerman

Examples

```
d <- data.frame(X=runif(20), Y=runif(20))
plot(d, xlim=c(0,1), ylim=c(0,1))

# draw rug of ticks on each axis where the coordinates are
rug_multicolor(x=d$X, col_vec=colorspace::rainbow_hcl(n=20), ticksize=-0.05)
rug_multicolor(x=d$Y, plot_side=4, col_vec=colorspace::rainbow_hcl(n=20), ticksize=-0.05)
```

`shark_data_longlat` *Raw shark data spline-interpolated to 90-second intervals*

Description

Raw shark data spline-interpolated to 90-second intervals (matrix)

Usage

```
data(shark_data_longlat)
```

Format

The format is: int [1:226400, 1:7] 1217951746 1217951836 1217951926 1217952016 1217952106 1217952196 1217952286 1217952376 1217952466 1217952556 ... - attr(*, "dimnames")=List of 2 ..$: NULL ..$: chr [1:7] "date_as_sec" "lat" "lon" "t_intervals" ...

Source

Espinoza, Mario, Farrugia, Thomas J., and Christopher G. Lowe. Habitat use, movements and site fidelity of the gray smooth-hound shark in a newly restored Southern California estuary. Journal of Experimental Marine Biology and Ecology, 2011.

`shark_data_raw` *Original shark data*

Description

Original shark data observations, unequally spaced in time

Usage

```
data(shark_data_raw)
```

Format

A data frame with 68528 observations on the following 12 variables.

`tag` a factor with levels `GSH01` `GSH02` `GSH03` `GSH04` `GSH05` `GSH06` `GSH07` `GSH08` `GSH09` `GSH10` `GSH11` `GSH12` `GSH13` `GSH14` `GSH15` `GSH16` `GSH17` `GSH18` `GSH19` `GSH20` `GSH21` `GSH22`

`X` a numeric vector

`Y` a numeric vector

`logvelocity` a numeric vector

`bearing.to.east.tonext.rad` a numeric vector

`turn.angle.rad` a numeric vector

`state.guess2` a numeric vector

`prev.guess2` a numeric vector

`time_to_next` a numeric vector

`dx_to_next` a numeric vector

`dy_to_next` a numeric vector

`date_as_sec` a numeric vector

Source

Espinoza, Mario, Farrugia, Thomas J., and Christopher G. Lowe. Habitat use, movements and site fidelity of the gray smooth-hound shark in a newly restored Southern California estuary. Journal of Experimental Marine Biology and Ecology, 2011.

Examples

```
##stored as separate integer and numeric variables for storage purposes

data(shark_data_raw, package="animalEKF")
shark_data <- do.call(cbind, shark_data_raw)
head(shark_data)
```

`shark_vis_longlat` *Shiny app for visualizing observed shark movement.*

Description

Shiny app for visualizing observed shark movement.

Usage

```
shark_vis_longlat()
```

Details

This shiny app visually illustrates movement of sharks in the dataset referenced in the paper below. The observations to be visualized are selected in "Range of observed steps". They are then linearly interpolated with step size "seconds to interpolate". In our paper, we model the impact of other sharks' behaviors in a spatial-temporal neighborhood. If desired, a spatial neighborhood of desired size "spatial radius" will appear around each shark in the presence of other sharks. It will flash red if another shark enters in that radius (i.e., they are neighbors).

Source

Espinoza, Mario, Farrugia, Thomas J., and Christopher G. Lowe. Habitat use, movements and site fidelity of the gray smooth-hound shark in a newly restored Southern California estuary. Journal of Experimental Marine Biology and Ecology, 2011.

`sim_trajectory_joint` *Simulation and interpolation of trajectories.*

Description

`sim_trajectory_joint` simulates regular-step trajectories under correlated random walk (CRW). `interp_trajectory_joint` interpolates regular steps to irregular ones drawn from a log-normal distribution.

Usage

```
sim_trajectory_joint(area_map=NULL, centroids=matrix(c(0,0), ncol=2),
                     transition_matrices=list(matrix(c(10,3,2,9),
                     ncol=2, byrow=TRUE)),
                     mu0_pars=list(alpha=c(-4 ,-1.6), beta=c(0,0)),
                     var0_pars=list(alpha=c(1.6,0.16), beta=c(2,0.5)),
                     N=100, nstates=2, reg_dt=120, gen_irreg=TRUE,
                     one_d=FALSE, dt_lnorm_mu=log(120), dt_lnorm_sd=1,
                     dt_vals=NULL, starting_polygon=area_map,
                     nsharks=1, interact=FALSE,
                     interact_pars=list(interacting_sharks=c(1:nsharks),
                     time_radius=60*30, spat_radius=200, min_num_neibs=10,
                     eta_mu=c(2,1), rho_sd=c(0.75, 0.75)),
                     time_dep_trans=FALSE, trans_alpha=c(1, 1.5))

interp_trajectory_joint(d, nstates, one_d, dt_lnorm_mu=5, dt_lnorm_sd=1,
                        dt_vals=NULL, centroids=matrix(c(0,0), ncol=2))
```

Arguments

`area_map`	Shapefile within which the observations are located (optional). Should be the output of applying `sf::st_geometry` on an object of class `sf`. If input is NULL, a default rectangular shapefile is created.
`centroids`	Matrix with two columns specifying the centroids of regions. The number of rows specifies the number of regions.
`transition_matrices`	A list of 2x2 matrices specifying the Dirichlet parameters for behavior transition probabilities. The list is replicated so it's the length of the number of regions. If `nstates`=1 then these are not used since there is only one behavior.
`mu0_pars`	List of mean values of alpha (=log-speed if 2-D, and velocity if 1-D) and beta (turn angle, ignored for 1-D) for one or two behavioral states.
`var0_pars`	List of variances of alpha and beta distributions (see `mu0_pars`).
`N`	Number of regular steps to simulate.
`nstates`	Number of behavioral states. For now restricted to a maximum of 2.
`reg_dt`	Length in seconds of each regular interval.
`gen_irreg`	Logical. If TRUE, then use `interp_trajectory_joint` to make irregular steps.
`one_d`	Logical. If TRUE, then simulation occurs on 1-D line, if FALSE (the default) it is 2-D.
`dt_lnorm_mu`	Mean parameter mu of the log-normal distribution to draw time step lengths.
`dt_lnorm_sd`	Standard deviation parameter sigma of the log-normal distribution to draw time step lengths.
`starting_polygon`	Polygon to draw starting coordinates in. This helps if you want the trajectories to start around the same area.
`nsharks`	Number of sharks to simulate trajectories for. If `nsharks`>1, then joint effects may take place.
`interact`	Logical. If TRUE, simulate interaction parameters of neighborhood (either 1-D or 2-D). If `nstates`=1, automatically set to FALSE.
`interact_pars`	List of interaction priors: 1) `interacting_sharks` means which of the sharks 1...`nsharks` are to use interaction parameters; 2) `time_radius` is the time in seconds, and 3) `spat_radius` is the spatial radius is meters to consider for spatial neighbors; 4) `min_num_neibs` is the minimum number of time and spatial radius observations that need to exist to constitute a neighborhood; 5) `eta_mu` is the vector of mean value for the interaction parameter `eta`; `rho_sd` is the vector of standard deviations of the interaction multiplier rho.
`time_dep_trans`	Logical. If TRUE, state transition matrices are time-dependent meaning that probability depends on the number of steps a shark has remained in the current state.
`trans_alpha`	If `time_dep_trans`=TRUE, the transition alpha parameters for the Dirichlet distribution for drawing behaviors.
`d`	Input for `interp_trajectory_joint`. An array, usually output by `sim_trajectory_joint`, of regular-step trajectories.

`dt_vals`	An optional vector of time difference values. By default is NULL, meaning time gaps will be generated by `dt_lnorm_mu` and `dt_lnorm_sd`, but supplying a vector to `dt_vals` lets the user specify the time gaps rather than having them be randomly generated.

Value

`d`	Array of regular-step trajectory locations.
`d_ds`	Object `d` in format `data.frame`.
`di`	If `gen_irreg==TRUE`, is the non-constant step length locations.

Author(s)

Samuel Ackerman

Examples

```
# read shapefile and convert into a Polygon
bolsachica <- sf::st_read(system.file("shapes/FTB_lines.shp", package="animalEKF")[1])
bolsachica <- sf::st_polygonize(bolsachica)

island <- sf::st_read(system.file("shapes/FTB_island.shp", package="animalEKF")[1])
# the actual available room for movement is the area in the water, subtracting the island inside
bolsachica <- sf::st_difference(sf::st_geometry(bolsachica), sf::st_geometry(island))


# sample 5 points approximately equally spaced within the shapefile, as region centroids
regions <- sf::st_sample(x=sf::st_geometry(bolsachica), size=5, type="regular", exact=TRUE)
# extract the coordinates
regions <- as.data.frame(sf::st_coordinates(regions)[, c("X","Y")])

#define Voronoi tessellation tile in which to start shark paths
vortess <- deldir::deldir(x=regions[,"X"], y=regions[,"Y"], wlines="tess",
  plotit=FALSE, suppressMsge=TRUE)
# convert these to a set of Polygons, and choose one of them as the starting polygon
vtiles <- sf::st_as_sf(tess2spat(vortess))
sf::st_crs(vtiles) <- sf::st_crs(bolsachica)
# extract only the 3rd tile
# note, want to have the simulation paths spread out, so a given draw may result in
# cramped and thus hard to estimate paths

starting_polygon <- sf::st_sfc(vtiles[[1]][[3]], crs=sf::st_crs(vtiles))
starting_polygon <- sf::st_intersection(sf::st_geometry(bolsachica),
sf::st_geometry(starting_polygon))

#define list of transition matrices between behaviors

tmat_list <- list(matrix(c(8, 2, 2, 4), ncol=2, byrow=TRUE),
                  matrix(c(1.5*5, 1.5*1, 3, 3), ncol=2, byrow=TRUE),
                  matrix(c(7, 1, 1, 7), ncol=2, byrow=TRUE))

#generate 4-shark simulated trajectory with 100 regular steps of length 120 seconds.
#Sharks 3 and 4 will be interacting with the others, but 1 and 2 will not.

nsharks <- 4

#simulate trajectory
#setting gen_irreg=TRUE generates an irregular trajectory from the regular-step one
#with the log-normal specified in dt_lnorm_mu and dt_lnorm_sd
#sim_4sharks$di would contain the irregular dataset
#otherwise, say you wanted to try different interpolations, you can use the same regular
#step from sim_trajectory_joint and then interpolate separately with interp_trajectory_joint.

#make simulated trajectories all start in the same area so they will be close enough to be
#interacting, for the purposes of this exercise
#note that the simulation may time out trying to draw points in this starting polygon that end
#up in the shapefile boundary

nsteps_sim <- 100
reg_dt <- 120



sim_4sharks <- sim_trajectory_joint(area_map=sf::st_geometry(bolsachica), centroids=regions,
                                    transition_matrices=tmat_list, nsharks=nsharks,
                                    mu0_pars=list(alpha=c(-4 ,-1.6), beta=c(0,0)),
                                    var0_pars=list(alpha=c(1,0.25), beta=c(1,.25)),
                                    N=nsteps_sim, nstates=2, reg_dt=reg_dt,
                                    gen_irreg=FALSE, one_d=FALSE,
                                    starting_polygon=starting_polygon, interact=TRUE,
                                    interact_pars=list(interacting_sharks=c(3:4),
                                    time_radius=60*30, spat_radius=150,
                                    min_num_neibs=10,
                                    eta_mu=c(2,1), rho_sd=c(0.75, 0.75)),
                                    time_dep_trans=FALSE,
                                    dt_lnorm_mu=log(120), dt_lnorm_sd=0.4)




#plot trajectories

shark_names <- dimnames(sim_4sharks$d)[[ 3 ]]
shark_colors <- 2:5
names(shark_colors) <- shark_names


sp::plot(bolsachica, main="Full trajectories")
deldir::plot.deldir(vortess, wlines="tess", add=TRUE)

for (ss in shark_names) {
    lines(sim_4sharks$d[,c("X","Y"), ss], col=shark_colors[ss])
```

```
}

#now interpolate to uneven steps with lognormal mean log(120) (so they are on
#average the same as the regular steps and sd=0.4
#d is the regular step, di is irregular

#if want to interpolate separately.  Otherwise just set gen_irreg=TRUE above
#this is so you can interpolate a dataset not generated by sim_trajectory_joint
#if gen_irreg=TRUE in sim_trajectory_joint,
#interp_ds will be returned as the 'di' object

interp_ds <- interp_trajectory_joint(d=sim_4sharks$d, nstates=2,
                                     one_d=FALSE,
                                     dt_lnorm_mu=log(reg_dt),
                                     dt_lnorm_sd=0.4,
                                     centroids=regions)



#now plot observed ones, may differ
sp::plot(bolsachica, main="Observed trajectories")
deldir::plot.deldir(vortess, wlines="tess", add=TRUE)

for (ss in shark_names) {
    lines(interp_ds[ interp_ds$tag == ss ,c("X","Y")], col=shark_colors[ss])
}




#try to recover EKF with steps at the original 120 seconds
#use the original simulated transition and foraging probabilities for comparison


#intial values for some parameters
tau_pars_init <- c(8, 14, 10,1) #2
sigma_pars_init <- c(5, 8, 8, 3)

#measurement error

bmat <- matrix(c(1, -0.3, -0.3, 1), ncol=2)
Errvar_init1 <-5*20*bmat
Errvar_init2 <- 15*20*bmat

#particle error
Particle_err_init <- 0.5*20*bmat

# only estimate movement on first 5 steps
# for better results, npart should be set higher, like 150 or more
nsteps_estimate <- 5
npart <- 15
```

```
#again, if you use gen_irreg=TRUE in sim_trajectory_joint,
#the input 'd' argument should be sim_4sharks$di or interp_ds

#NOTE: user should set output_plot=TRUE to see PDF,
#for purposes of package testing we set it to FALSE
# if FALSE, plots will still appear in the console


ekf_interp_mod <- EKF_interp_joint(d=interp_ds, npart=npart,
                                   area_map=bolsachica,
                                   state_favor=c(1,2),
                                   centroids=regions,
                                   sigma_pars=sigma_pars_init,
                                   tau_pars=tau_pars_init,
                                   Errvar0=list(Errvar_init1, Errvar_init2),
                                   Particle_errvar0=Particle_err_init,
                                   mu0_pars=list(alpha=c(-4 ,-1.3), beta=c(0,0)),
                                   truncate=TRUE,
                                   neff_sample=0.75, dirichlet_init=c(8,2,2,4),
                                   smoothing=TRUE, fix_smoothed_behaviors=FALSE,
                                   time_dep_trans=FALSE, resamp_full_hist=FALSE,
                                   nstates=2, reg_dt=reg_dt, interact=TRUE,
                                   maxStep=nsteps_estimate, update_eachstep=TRUE,
                                   compare_with_known=TRUE,
                                   known_trans_prob=sim_4sharks$true_transition_prob,
                                   known_foraging_prob=sim_4sharks$true_foraging_prob,
                                   known_regular_step_ds=sim_4sharks$d_ds,
                                   output_plot=FALSE)


#simulate one-dimensional movement for 1 robot (shark)
#here we use gen_irreg=TRUE instead of generating a separate interpolation object

one_d <- sim_trajectory_joint(centroids=NULL, N=nsteps_sim,
                              mu0_pars=list(alpha=c(4, 9)),
                              var0_pars=list(alpha=c(1, 1)),
                              transition_matrices=tmat_list[[ 1 ]], nstates=2,
                              reg_dt=reg_dt, gen_irreg=TRUE, one_d=TRUE,
                              dt_lnorm_mu=log(120), dt_lnorm_sd=0.55)




#measurement error
bmat <- matrix(1)
Errvar_init1 <-1*bmat
Errvar_init2 <-3*bmat


#particle error
Particle_err_init <- 0.1*bmat
```

```
ekf_1d <- EKF_1d_interp_joint(d=one_d$di, npart=npart,  maxStep=nsteps_estimate,
                              state_favor=c(1,1), nstates=2, lowvarsample=TRUE,
                              neff_sample=1, time_dep_trans=FALSE, reg_dt=reg_dt,
                              max_int_wo_obs=15, resamp_full_hist=FALSE,
                              alpha0_pars=list(mu0=c(4, 9), V0=c(0.25, 0.25)),
                              sigma_pars=sigma_pars_init,
                              Errvar0=list(Errvar_init1, Errvar_init2),
                              Particle_errvar0=Particle_err_init,
                              compare_with_known=TRUE,
                              known_trans_prob=one_d$true_transition_prob,
                              known_foraging_prob=one_d$true_foraging_prob,
                              known_regular_step_ds=one_d$d_ds, update_eachstep=TRUE,
                              smoothing=TRUE, output_plot=FALSE)
```

spline_interp *Bezier spline interpolation of observations.*

Description

Calculate a Bezier spline interpolation of irregular observations to regular-length time intervals.

Usage

```
spline_interp(di, area_map=NULL, t_reg=NULL, reg_dt=120,
                max_dt_wo_obs=60*30, maxStep=NULL,
                centroids=matrix(c(0,0), ncol=2),
                nstates=2, spline_deg=3, split_logv=-3)
```

Arguments

di	Object of class `data.frame` containing irregular-spaced observations. Dataset must contain the following fields: 1) `"X"` and `"Y"`: X and Y location coordinates, 2) `"date_as_sec"`: time/date of observation as seconds since an epoch. `"time_to_next"` should also be included, but will be calculated if not. Also `"tag"` (animal identifier) should be included as well. If not, all observations are assumed to be of the same animal. This can be generated from `sim_trajectory_joint` with `gen_irreg=TRUE`.
area_map	Shapefile that all interpolated points should be inside of.
t_reg	Desired time steps (must have a constant difference) to interpolate to. If is given, the default value of `reg_dt` is overridden. Will be truncated to the set of values within the range of observed values of `di$date_as_sec`.
reg_dt	Length in seconds of each regular interval.

`max_dt_wo_obs`	When interpolating, the maximum time length without observations for a given shark that we will interpolate. If this is exceeded, algorithm will wait until next observation and start from there.
`maxStep`	Maximum number of regular steps to interpolate.
`centroids`	Matrix with two columns specifying the centroids of regions. If `NULL`, only one region will be used.
`nstates`	Number of behavioral states. For now restricted to a maximum of 2.
`spline_deg`	Degree of spline. The default is 3, or a cubic. Every `spline_deg+1` observations will be used to construct one spline segment.
`split_logv`	If `nstates=2`, state 1 from the interpolated values will be designated by the logvelocity being < `split_logv`.

Value

`d`	Array of regular step locations.
`di`	Original irregular-step dataset.
`shark_names`	Vector of the names of sharks in the dataset.
`d_ds`	Output regular-step dataset `d` in form `data.frame`.

Author(s)

Samuel Ackerman

References

Bezier R package. Aaron Olsen.

Examples

```
#can also be 'di' output of sim_trajectory_joint (set gen_irreg=TRUE)

di <- data.frame(X=runif(n=9), Y=runif(n=9),
                 time_to_next=c(2,4,15,8,5,18,3,5,NA))
di$date_as_sec <- c(0, cumsum(di$time_to_next[-9]))
region_centroids <- cbind(X=runif(2), Y=runif(2))

#one log observation with dt =18 > 16 will be omitted
spl <- spline_interp(di=di, area_map=NULL, reg_dt=3, max_dt_wo_obs=16, maxStep=NULL,
                     centroids=region_centroids, nstates=2, spline_deg=3, split_logv=-3)

plot(di[,c("X","Y")], xlim=c(0,1), ylim=c(0,1), type="b", las=1,
     "Observations interpolated by regular interval spline")
lines(spl$d_ds[,c("X","Y")], type="l", col="red")
legend("topleft", col=1:2, legend=c("observations","spline"), lty=1)
```

tess2spat *Convert Voronoi tessellation tiles to a shapefile.*

Description

Convert Voronoi tessellation tiles to a shapefile.

Usage

```
tess2spat(obj, idvec=NULL)
```

Arguments

`obj`	Voronoi tessellation object created through function `deldir`.
`idvec`	Optional vector of ids for output shapefile polygons.

Value

Object of class `SpatialPolygons`.

Author(s)

Samuel Ackerman

Examples

```
library(deldir)
library(sp)

vortess <- deldir(x=runif(8), y=runif(8), plotit=FALSE, suppressMsge=TRUE)
old_pars <- par(mfcol=par()$mfcol)

par(mfcol=c(1,2))
deldir::plot.deldir(vortess, wlines="tess", xlim=c(0,1), ylim=c(0,1))
vortess_shape <- tess2spat(obj=vortess)
plot(vortess_shape)
par(old_pars)
```

Index