Package ‘bSims’

May 21, 2024

Type Package

Title Agent-Based Bird Point Count Simulator

Version 0.3-2

Date 2024-05-21

Author Peter Solymos [aut, cre] (<https://orcid.org/0000-0001-7337-1740>)

Maintainer Peter Solymos <psolymos@gmail.com>

Description A highly scientific and utterly addictive
bird point count simulator
to test statistical assumptions, aid survey design,
and have fun while doing it (Solymos 2024 <doi:10.1007/s42977-023-00183-2>).
The simulations follow time-removal and distance sampling models
based on Matsuoka et al. (2012) <doi:10.1525/auk.2012.11190>,
Solymos et al. (2013) <doi:10.1111/2041-210X.12106>,
and Solymos et al. (2018) <doi:10.1650/CONDOR-18-32.1>,
and sound attenuation experiments by
Yip et al. (2017) <doi:10.1650/CONDOR-16-93.1>.

License GPL-2

LazyLoad yes

Depends intrval, mefa4, MASS, deldir (>= 1.0-2)

Imports parallel, pbapply

URL https://github.com/psolymos/bSims

BugReports https://github.com/psolymos/bSims/issues

Suggests knitr, rmarkdown, detect, shiny

VignetteBuilder knitr

Language en-US

NeedsCompilation no

Repository CRAN

Date/Publication 2024-05-21 10:40:20 UTC

R topics documented:

`acceptreject` *Spatial point process simulator*

Description

Spatial point process simulator based on accept/reject algorithm.

Usage

```
acceptreject(n, f = NULL, x0 = 0, x1 = 1, y0 = 0, y1 = 1,
  m = 0, maxit = 100, fail = FALSE)
```

Arguments

`n`	number of points to generate.
`f`	a function returning probability (value between 0 and 1) given distance as the first and only argument. The function generates spatially uniform Poisson point process (complete spatial randomness) when `NULL`.
`x0, x1, y0, y1`	x and y ranges (bounding box).
`m`	margin width for avoiding edge effects.
`maxit`	maximum number of iterations per point to try if no acceptance happens.
`fail`	logical, what to do when there is a problem. `TRUE` gives error, the default `FALSE` gives only a warning.

Value

A matrix with `n` rows and 2 columns for x and y coordinates.

Author(s)

Peter Solymos

Examples

```
## complete spatial randomness
plot(acceptreject(100), asp=1)

## more systematic
distance <- seq(0,1,0.01)
f <- function(d)
  (1-exp(-d^2/0.1^2) + dlnorm(d, 0.2)/dlnorm(exp(0.2-1),0.2)) / 2
op <- par(mfrow = c(1, 2))
plot(distance, f(distance), type="l")
plot(acceptreject(100, f, m=1), asp=1)
par(op)

## more clustered
f <- function(d)
  exp(-d^2/0.1^2) + 0.5*(1-exp(-d^2/0.4^2))
op <- par(mfrow = c(1, 2))
plot(distance, f(distance), type="l")
plot(acceptreject(100, f, m=1), asp=1)
par(op)
```

`bsims_init` *bSims simulation functions*

Description

Functions to initialize, populate, animate, detect, and transcribe simulated birds in a point count.

Usage

```
bsims_init(extent = 10, road = 0, edge = 0, offset = 0)

bsims_populate(x, density = 1, abund_fun = NULL, xy_fun = NULL,
  margin = 0, maxit = 100, fail = FALSE, ...)

bsims_animate(x, vocal_rate = 1, move_rate = 0, duration = 10,
  movement = 0, mixture = 1, avoid = c("none", "R", "ER"),
  initial_location=FALSE, allow_overlap=TRUE, ...)

bsims_detect(x, xy = c(0, 0), tau = 1, dist_fun = NULL,
  event_type = c("vocal", "move", "both"),
  sensitivity=1, direction=FALSE, ...)

bsims_transcribe(x, tint = NULL, rint = Inf,
  error = 0, bias = 1,
  condition=c("event1", "det1", "alldet"),
  event_type=NULL, perception=NULL, ...)
```

```
bsims_all(...)

## S3 method for class 'bsims_landscape'
print(x, ...)
## S3 method for class 'bsims_population'
print(x, ...)
## S3 method for class 'bsims_events'
print(x, ...)
## S3 method for class 'bsims_detections'
print(x, ...)
## S3 method for class 'bsims_transcript'
print(x, ...)
## S3 method for class 'bsims_all'
print(x, ...)
```

Arguments

`extent`	extent of simulation area, an extent x extent square with (0,0) at the center.
`road`	half width of the road stratum (perpendicular to the y axis).
`edge`	width of edge, same width on both sides of the road stratum.
`offset`	offset to apply to road and edge strata relative to the center in the x direction.
`x`	a simulation object.
`density`	population density, D, recycled 3x for the 3 strata (H: habitat, E: edge, R: road).
`abund_fun`	function to simulate abundance, N ~ Poisson(lambda), lambda=DA by default.
`xy_fun`	function used to simulate nest locations, see `acceptreject`. The function should return probability (value between 0 and 1), `NULL` means complete spatial randomness.
`margin, maxit, fail`	arguments passed to `acceptreject` when using `xy_fun` to simulate nest locations.
`vocal_rate, move_rate`	Vocal and movement rates (see `events`). Both of these rates can be: a single number; a vector of length `length(mixture)` (behavior based finite mixture groups); a vector of length 3 with `mixture=1` (corresponding to HER strata); or a matrix of dimension 3 x `length(mixture)` (HER strata x number of behavior based groups).
`duration`	total time duration to consider (in minutes), passed to `events`.
`movement`	standard deviation for a bivariate Normal kernel to simulate locations centered at the nest location, passed to `events`. Can refer to the same stratum and behavior based groups as `move_rate`.
`mixture`	behavior based finite mixture group proportions.
`avoid`	range along the x axis to avoid with respect to movement locations, passed to `events`.
`initial_location`	logical, `move_rate` and `vocal_rate` are silently ignored if `TRUE` and nest locations are provided as part of the events table. This renders all individuals equally available for detection.

`allow_overlap`	logical, allowing overlap between neighboring nests when movement is involved. If `FALSE`, Voronoi tessellation is used to prevent overlap. If `TRUE`, the unconstrained bivariate Normal kernel is used.
`xy`	a vector of x and y coordinates describing the position of the observer.
`tau`	parameter of the distance function. Can be a single numeric value; a vector of length 2 to provide parameters for vocalization (1st value) and movement (2nd value) related events; (H: habitat, E: edge, R: road, in this order); a vector of length 3 to provide parameters for the 3 strata (H: habitat, E: edge, R: road); or a 3 x 2 matrix combining strata (rows) and vocalization/movement (columns) related parameters. Segmented sound attenuation is used when the values are different in the 3 strata (see `dist_fun2`).
`dist_fun`	distance function (1st argument is distance, second is `tau`).
`event_type`	type of events to access (vocal, movement, or both). Inherits value from input object when `NULL`.
`tint`	time interval break points in minutes.
`rint`	distance interval break points in units of 100 meter.
`condition`	conditioning type to define availability for each individual: `"event1"`: the 1st event (detected or not); `"det1"`: the 1st detection; `"alldet"`: all detections (counting the same individual multiple times).
`error`	log scale standard deviation (SD) for distance estimation error, see `rlnorm2`. When `direction=TRUE`, error changes based on the angle between the observer and the individual's (random) singing direction. When the bird faces the observer (0 degrees) SD is 0, when the bird is facing away (180 degrees) SD is `error`. In the range between 0-180 degrees the SD is changing according to the cosine of the degree: SD*(0.5-cos(degree*pi/180)/2).
`bias`	nonnegative numeric, the distance estimation bias. The default value (1) means no bias, <1 indicates negative bias (perceived distance is less than true distance), >1 indicates positive bias (perceived distance is higher than true distance). This acts as a multiplier and can be combined with `error`. When `direction=TRUE`, bias changes based on the angle between the observer and the individual's (random) singing direction. When the bird faces the observer (0 degrees) perceived distance equals the true distance, when the bird is facing away (180 degrees) perceived distance is `bias` * true distance. In the range between 0-180 degrees the bias is changing according to the cosine of the degree: 1+(bias-1)*(0.5-cos(degree*pi/180)/2).
`perception`	perceived number of individuals relative to the actual number of individuals. A non-negative number (<1 values lead to under counting, >1 values lead to over counting), or `NULL` (observer correctly identifies all individuals).
`sensitivity`	non-negative numeric value indicating the sensitivity of the sensor receiving the signal. Can be of length 1 (applies to both vocal and movement events) or a named vector of length 2 (names should indicate which one is `"vocal"` or `"move"`). Sensitivity of 1 means that the process captured by `tau` is unaffected. Less than 1 values indicate lower sensitivity (effectively decreasing tau), larger than 1 values indicate higher sensitivity (effectively increasing tau).

direction
: logical. When TRUE, tau for vocalizations (not for movement) changes based on the angle between the observer and the individual's (random) singing direction. When the bird faces the observer (0 degrees) tau is unaffected, when the bird is facing away (180 degrees) tau is sensitivity * tau. In the range between 0-180 degrees the effect is changing according to the cosine of the degree (0.5-cos(degree*pi/180)/2).

...
: other arguments passed to underlying functions. For the bsims_all wrapper, it means all the arguments (except for x) that the underlying bsims_* functions have. bsims_all can also take a single list as its argument.

Details

The functions capturing the simulation layers are supposed to be called in sequence, allowing to simulate multiple realities by keeping preceding layers intact. Construction allows easy piping. The bsims_all function is a wrapper for the bsims_* layer functions.

The simulations follow time-removal and distance sampling models based on Matsuoka et al. (2012) <doi:10.1525/auk.2012.11190>, Solymos et al. (2013) <doi:10.1111/2041-210X.12106>, and Solymos et al. (2018) <doi:10.1650/CONDOR-18-32.1>, and sound attenuation experiments by Yip et al. (2017) <doi:10.1650/CONDOR-16-93.1>.

Value

bsims_init returns a landscape object.

bsims_populate returns a population object.

bsims_animate returns an events object.

bsims_detect returns a detections object.

bsims_transcribe returns a transcript object.

get_table returns the removal table.

bsims_all returns a closure with $settings(), $new(recover = FALSE), and $replicate(B, recover = FALSE, cl = NULL) functions. The settings function returns the input arguments as a list; the new function returns a single transcript object; the replicate function takes an argument for the number of replicates (B) and returns a list of transcript objects with B elements. The cl argument is used to parallelize the work, can be a numeric value on Unix/Linux/OSX, or a cluster object on any OS, see examples. The 'recover = TRUE' argument allows to run simulations with error catching based on try.

Note that simulated objects returned by bsims_all will contain different realizations and all the conditionally independent layers. Use a layered approach if former layers are meant to be kept identical across runs.

Author(s)

Peter Solymos

References

Matsuoka, S. M., Bayne, E. M., Solymos, P., Fontaine, P., Cumming, S. G., Schmiegelow, F. K. A., & Song, S. A., 2012. Using binomial distance-sampling models to estimate the effective detection radius of point-counts surveys across boreal Canada. *Auk*, **129**: 268–282. <doi:10.1525/auk.2012.11190>

Solymos, P., Matsuoka, S. M., Bayne, E. M., Lele, S. R., Fontaine, P., Cumming, S. G., Stralberg, D., Schmiegelow, F. K. A. & Song, S. J., 2013. Calibrating indices of avian density from non-standardized survey data: making the most of a messy situation. *Methods in Ecology and Evolution*, **4**: 1047–1058. <doi:10.1111/2041-210X.12106>

Solymos, P., Matsuoka, S. M., Cumming, S. G., Stralberg, D., Fontaine, P., Schmiegelow, F. K. A., Song, S. J., and Bayne, E. M., 2018. Evaluating time-removal models for estimating availability of boreal birds during point-count surveys: sample size requirements and model complexity. *Condor*, **120**: 765–786. <doi:10.1650/CONDOR-18-32.1>

Yip, D. A., Bayne, E. M., Solymos, P., Campbell, J., and Proppe, J. D., 2017. Sound attenuation in forested and roadside environments: implications for avian point count surveys. *Condor*, **119**: 73–84. <doi:10.1650/CONDOR-16-93.1>

See Also

Plotting functions: `plot.bsims_landscape`

Getter functions: `get_nests`, `get_events`, `get_detections`, `get_abundance`, `get_density` `get_table`

Shiny apps: `run_app`

`acceptreject`, `events`, `estimate`

Examples

```
phi <- 0.5
tau <- 1:3
dur <- 10
rbr <- c(0.5, 1, 1.5, Inf)
tbr <- c(3, 5, 10)
(l <- bsims_init(10, 0.5, 1))
(p <- bsims_populate(l, 1))
(a <- bsims_animate(p, vocal_rate=phi, duration=dur))
(o <- bsims_detect(a, tau=tau))
(x <- bsims_transcribe(o, tint=tbr, rint=rbr))

plot(x)
get_table(x, "removal")
get_table(x, "visits")

head(get_events(a))
plot(get_events(a))

head(get_detections(o))
plot(get_detections(o), "time")
plot(get_detections(o), "distance")

## wrapper function for all the bsims_* layers
```

```
b <- bsims_all(road=1, density=0.5, tint=tbr, rint=rbr)
## alternatively: supply a list
#settings <- list(road=1, density=0.5, tint=tbr, rint=rbr)
#b <- bsims_all(settings)
b$settings()
b$new()
bb <- b$replicate(3)
lapply(bb, get_table)


## parallel simulations
library(parallel)
b <- bsims_all(density=0.5)
B <- 4  # number of runs
nc <- 2 # number of cores
## sequential
system.time(bb <- b$replicate(B, cl=NULL))
## parallel clusters
cl <- makeCluster(nc)
## note: loading the package is optional
system.time(clusterEvalQ(cl, library(bSims)))
system.time(bb <- b$replicate(B, cl=cl))
stopCluster(cl)
## parallel forking
if (.Platform$OS.type != "windows") {
  system.time(bb <- b$replicate(B, cl=nc))
}
```

---

dist_fun2 *Distance function with segmented attenuation*

Description

Distance function with segmented attenuation crossing a number of boundaries of strata with different attenuation characteristics following results in Yip et al. (2017).

Usage

```
dist_fun2(d, tau, dist_fun, breaks = numeric(0), ...)
```

Arguments

d	distance from observer.
tau	a parameter passed to the the distance function. Length of tau must equal length(b) + 1 referring to distance function parameters in the different strata (a stratum is defined by an interval surrounded by 1 or 2 boundaries).
dist_fun	distance function taking two arguments: distance, and tau, see examples.

breaks — distance breakpoints, must be `length(tau) - 1` in length. These breakpoints represent the boundaries between the intervals characterized by homogeneous attenuation characteristics.

... — other arguments passed to `dist_fun`

Value

Probability of detection given the distance, stratum specific parameters and the arrangement of breakpoints.

Author(s)

Peter Solymos

References

Yip, D. A., Bayne, E. M., Solymos, P., Campbell, J., and Proppe, J. D., 2017. Sound attenuation in forested and roadside environments: implications for avian point count surveys. *Condor*, **119**: 73–84. <doi:10.1650/CONDOR-16-93.1>

Examples

```
tau <- c(1, 2, 3, 2, 1)
d <- seq(0, 4, 0.01)
dist_fun <- function(d, tau) exp(-(d/tau)^2) # half normal
#dist_fun <- function(d, tau) exp(-d/tau) # exponential
#dist_fun <- function(d, tau) 1-exp(-(d/tau)^-2) # hazard rate

b <- c(0.5, 1, 1.5, 2) #  boundaries

op <- par(mfrow=c(2, 1))
plot(d, dist_fun2(d, tau[1], dist_fun), type="n",
  ylab="P(detection)", xlab="Distance", axes=FALSE,
  main="Sound travels from left to right")
axis(1)
axis(2)
for (i in seq_len(length(b)+1)) {
  x1 <- c(0, b, 4)[i]
  x2 <- c(0, b, 4)[i+1]
  polygon(c(0, b, 4)[c(i, i, i+1, i+1)], c(0, 1, 1, 0),
    border=NA,
    col=c("darkolivegreen1", "burlywood1", "lightgrey",
    "burlywood1", "darkolivegreen1")[i])
}
lines(d, dist_fun2(d, tau[1], dist_fun))
lines(d, dist_fun2(d, tau[2], dist_fun))
lines(d, dist_fun2(d, tau[3], dist_fun))
lines(d, dist_fun2(d, tau, dist_fun, b), col=2, lwd=3)

plot(rev(d), dist_fun2(d, tau[1], dist_fun), type="n",
  ylab="P(detection)", xlab="Distance", axes=FALSE,
  main="Sound travels from right to left")
```

```
axis(1)
axis(2)
for (i in seq_len(length(b)+1)) {
  x1 <- c(0, b, 4)[i]
  x2 <- c(0, b, 4)[i+1]
  polygon(c(0, b, 4)[c(i, i, i+1, i+1)], c(0, 1, 1, 0),
    border=NA,
    col=c("darkolivegreen1", "burlywood1", "lightgrey",
    "burlywood1", "darkolivegreen1")[i])
}
lines(rev(d), dist_fun2(d, tau[1], dist_fun))
lines(rev(d), dist_fun2(d, tau[2], dist_fun))
lines(rev(d), dist_fun2(d, tau[3], dist_fun))
lines(rev(d), dist_fun2(d, tau, dist_fun, rev(4-b)), col=2, lwd=3)
par(op)
```

`estimate` *Estimate basic parameters*

Description

Estimate singing rates, effective distances, and density based on simulation objects using the QPAD approach (Solymos et al. 2013).

Usage

```
estimate(object, ...)
## S3 method for class 'bsims_transcript'
estimate(object, ...)
```

Arguments

`object`	simulation object.
`...`	other arguments passed to internal functions.

Details

The method evaluates removal design to estimate model parameters and density using the QPAD methodology using the 'detect' package.

The function only works with multiple time and distance intervals. It returns `NA` otherwise.

Value

A vector with values for singing rate (phi), effective detection distance (tau), and density.

Author(s)

Peter Solymos

References

Solymos, P., Matsuoka, S. M., Bayne, E. M., Lele, S. R., Fontaine, P., Cumming, S. G., Stralberg, D., Schmiegelow, F. K. A. & Song, S. J., 2013. Calibrating indices of avian density from non-standardized survey data: making the most of a messy situation. *Methods in Ecology and Evolution*, **4**: 1047–1058. <doi:10.1111/2041-210X.12106>

See Also

bsims_init

Examples

```
set.seed(2)
phi <- 0.5                 # singing rate
tau <- 1                   # EDR by strata
dur <- 10                  # simulation duration
tbr <- c(2, 4, 6, 8, 10)   # time intervals
rbr <- c(0.5, 1, 1.5, Inf) # counting radii

l <- bsims_init(10, 0.5, 1)# landscape
p <- bsims_populate(l, 10)  # population
e <- bsims_animate(p,       # events
  vocal_rate=phi, duration=dur)
d <- bsims_detect(e,        # detections
  tau=tau)
x <- bsims_transcribe(d,    # transcription
  tint=tbr, rint=rbr)

estimate(x)
```

---

events *Event time simulator*

Description

`timetoevent` turns exponential wait times to time-to-event data within a desired duration, it handles 0 and infinite rates in a robust manner. `events` simulates event times based on an exponential time-to-event distribution.

Usage

```
timetoevent(rate, duration)

events(vocal_rate = 1, move_rate = 1, duration = 10,
  movement = 0, avoid = c(0, 0))
```

Arguments

rate	rate for the exponential distribution (rexp).
duration	total time duration to consider (in minutes).
vocal_rate	vocal rate for exponential distribution (rexp), how often a vocal event happens per minute.
move_rate	movement rate for exponential distribution (rexp), how often a movement event happens per minute.
movement	standard deviation for a bivariate Normal kernel to simulate locations centered at the nest location.
avoid	range along the x axis to avoid with respect to movement locations, i.e. location for a movement event within this interval will be rejected and a new location drawn.

Value

An events object data frame with coordinates (x, y; centered at 0 that is nest location), event times (t) and indicator for vocal events (v).

Author(s)

Peter Solymos

See Also

rexp

Examples

```
timetoevent(0, 10)
timetoevent(Inf, 10)

rr <- 1
tt <- timetoevent(rr, 10)
op <- par(mfrow=c(1,2))
plot(ecdf(tt))
curve(1-exp(-rr*x), add=TRUE, col=2) # cdf

plot(stepfun(sort(tt), 0:length(tt)/length(tt)), ylab="F(x)")
curve(1-exp(-rr*x), add=TRUE, col=2) # cdf
par(op)

e <- events(movement=1, duration=60)
mx <- max(abs(e[,1:2]))
plot(e[,1:2], col="grey", type="l", asp=1,
  xlim=2*c(-mx, mx), ylim=2*c(-mx, mx))
points(e[,1:2], col=e$v+1)
abline(h=0, v=0, lty=2)
legend("topright", pch=21, col=1:2, horiz=TRUE,
  legend=c("movement", "vocalization"))
```

`expand_list` *Create a list from all combinations of arguments*

Description

Create a list from all combinations of the supplied vectors or lists.

Usage

```
expand_list(...)
```

Arguments

`...`	vectors or lists. All arguments must be named.

Value

A list containing one element for each combination of the supplied vectors and lists. The first factors vary fastest. The nested elements are labeled by the factors.

The function allows list elements to be vectors, functions, or `NULL`. If a vector element is supposed to be kept as a vector, use `list()`.

Author(s)

Peter Solymos

See Also

`expand.grid`

Examples

```
b <- expand_list(
  movement = c(0, 1, 2),
  rint = list(c(0.5, 1, 1.5, Inf)), # in a list to keep as one
  xy_fun = list(NULL, function(z) z))
b[[1]]
str(b)
```

get_nests *Access nests, events, detections, and totals*

Description

Access nests, events, detections, abundance, and density from simulation objects.

Usage

```
get_nests(x, ...)
## S3 method for class 'bsims_population'
get_nests(x, ...)

get_events(x, ...)
## S3 method for class 'bsims_events'
get_events(x, ...)

get_detections(x, ...)
## S3 method for class 'bsims_detections'
get_detections(x, ...)

get_abundance(x, ...)
## S3 method for class 'bsims_population'
get_abundance(x, ...)

get_density(x, ...)
## S3 method for class 'bsims_population'
get_density(x, ...)

get_table(x, ...)
## S3 method for class 'bsims_transcript'
get_table(x,
  type = c("removal", "visits"), ...)
```

Arguments

x	simulation object.
type	character, the type of table to return: "removal" includes only new individuals as time progresses, "visits" counts individuals in each time interval independent of each other.
...	other arguments passed to internal functions.

Details

get_nests extracts the nest locations.

get_events extracts the events.

get_detections extracts the detections.

get_abundance gets the realized total abundance (N), get_density gets the realized average density (abundance/area: N/A).

get_table returns the removal or visits table.

Value

get_abundance and get_density returns a non-negative numeric value.

get_nests returns a data frame with the following columns: i individual identifier, s spatial stratum (H: habitat, E: edge, R: road) x and y are coordinates of the nest locations, g is behavioral (mixture) group or NA.

get_events returns a data frame with the following columns: x and y are locations of the individual at the time of the event, t time of the event within the duration interval, v indicator variable for vocal (1) vs. movement (0) event, a direction for vocalization events (NA for movement) in degrees clockwise relative to north, i individual identifier.

get_detections returns a data frame with the following columns: x and y are locations of the individual at the time of the event, t time of the event within the duration interval, v indicator variable for vocal (1) vs. movement (0) event, a direction for vocalization events (NA for movement) in degrees clockwise relative to north, d distance from observer when detected (otherwise NA). f indicates the angle between the bird's vocalization direction (column a) relative to the observer (the value is 0 for movement events by default), i individual identifier, j perceived individual identifier.

get_table returns a matrix with distance bands as rows and time intervals as columns. The cell values are counts if the individuals detected in a removal fashion (only new individuals counter over the time periods) or in a multiple-visits fashion (counting of individuals restarts in every time interval).

Author(s)

Peter Solymos

See Also

bsims_init

Examples

```
phi <- 0.5                 # singing rate
tau <- 1:3                 # EDR by strata
dur <- 10                  # simulation duration
tbr <- c(3, 5, 10)         # time intervals
rbr <- c(0.5, 1, 1.5, Inf) # counting radii

l <- bsims_init(10, 0.5, 1)# landscape
p <- bsims_populate(l, 1)  # population
e <- bsims_animate(p,      # events
  vocal_rate=phi, duration=dur)
d <- bsims_detect(e,       # detections
  tau=tau)
x <- bsims_transcribe(d,   # transcription
```

```
  tint=tbr, rint=rbr)

## next locations
head(get_nests(p))
head(get_nests(e))
head(get_nests(d))
head(get_nests(x))

## abundance
get_abundance(p)
get_abundance(e)
get_abundance(d)
get_abundance(x)

## density
get_density(p)
get_density(e)
get_density(d)
get_density(x)

## events
head(get_events(e))
head(get_events(d))
head(get_events(x))

## detections
head(get_detections(d))
head(get_detections(x))

get_table(x, "removal")
get_table(x, "visits")
```

---

plot.bsims_landscape *Plot methods*

Description

Plot methods for different bSims objects.

Usage

```
## S3 method for class 'bsims_landscape'
plot(x,
  col_H, col_E, col_R,
  xlim = NULL, ylim = NULL, ...)

## S3 method for class 'bsims_population'
plot(x,
  pch_nest, col_nest, cex_nest, ...)
```

```
## S3 method for class 'bsims_events'
plot(x,
  event_type=c("vocal", "move", "both"), tlim = NULL,
  pch_nest, col_nest, cex_nest,
  pch_vocal, col_vocal, cex_vocal,
  lty_move, col_move, lwd_move, ...)

## S3 method for class 'bsims_detections'
plot(x,
  event_type=NULL, tlim = NULL,
  pch_nest, col_nest, cex_nest,
  pch_vocal, col_vocal, cex_vocal,
  lty_move, col_move, lwd_move,
  lty_det_vocal, col_det_vocal, lwd_det_vocal,
  lty_det_move, col_det_move, lwd_det_move,
  condition = "event1", ...)

## S3 method for class 'bsims_transcript'
plot(x,
  pch_nest, col_nest, cex_nest,
  pch_vocal, col_vocal, cex_vocal,
  lty_move, col_move, lwd_move,
  lty_det_vocal, col_det_vocal, lwd_det_vocal,
  lty_det_move, col_det_move, lwd_det_move,
  show_tint=TRUE, show_rint=TRUE,
  col_tint, col_rint, ...)

## S3 method for class 'bsims_events'
lines(x, tlim = NULL, ...)
## S3 method for class 'bsims_detections'
lines(x,
  event_type=NULL, tlim=NULL, condition="event1", ...)
## S3 method for class 'bsims_transcript'
lines(x,
  event_type=NULL, tlim=NULL, ...)

## S3 method for class 'bsims_population'
points(x, ...)
## S3 method for class 'bsims_events'
points(x,
  event_type=c("vocal", "move", "both"), tlim = NULL, ...)
## S3 method for class 'bsims_detections'
points(x,
  event_type=NULL, tlim=NULL, condition="event1", ...)

col2hex(col, alpha = FALSE)

## S3 method for class 'bsims_events_table'
```

```
plot(x,
  xlab, ylab, xlim, ylim, col_det_vocal, col_det_move, ...)
## S3 method for class 'bsims_detections_table'
plot(x,
  type=c("time", "distance"), xlab, ylab, xlim, ylim,
  col_det_vocal, col_det_move, ...)
```

Arguments

`x`	simulation object.
`col`	color values.
`col_H, col_E, col_R`	color values for the Habitat, Edge, and Road strata.
`event_type`	type of events to access. The value is inferred from the input object when `NULL`.
`xlim, ylim, tlim`	x, y, time intervals.
`xlab, ylab`	x and y axis labels.
`pch_nest, col_nest, cex_nest`	visual characteristics of nest locations.
`pch_vocal, col_vocal, cex_vocal`	visual characteristics of vocalization events.
`lty_move, col_move, lwd_move`	visual characteristics of movement events.
`lty_det_vocal, col_det_vocal, lwd_det_vocal`	visual characteristics of detection events related to vocalizations.
`lty_det_move, col_det_move, lwd_det_move`	visual characteristics of detection events related to movements.
`alpha`	alpha channel for colors.
`show_tint, show_rint`	whether time and distance intervals should be displayed.
`col_tint, col_rint`	colors for time and distance intervals.
`condition`	conditioning type to define availability for each individual, see `bsims_detect`.
`type`	what the x axis should be: time or distance.
`...`	other graphical arguments.

Details

The main plotting functions use a theme defined in the option `getOption("bsims_theme")`. Overriding these default settings allows customization.

Value

These plotting functions are called for their side effects and silently return the input object.

`col2hex` is modeled after `col2rgb` and returns a character vector giving hexadecimal color codes with or without alpha channel values.

Author(s)

Peter Solymos

See Also

bsims_init, col2rgb

Examples

```
b <- bsims_all(road=1, edge=2, move_rate=1, movement=0.2)$new()
o <- getOption("bsims_theme")
str(o)
n <- o
n$col_H <- "gold"
n$col_E <- "magenta"
n$col_R <- "black"
op <- par(mfrow=c(1, 2))
plot(b)
options("bsims_theme" = n) # apply new theme
plot(b)
par(op)
options("bsims_theme" = o) # reset old theme

col2hex(c(blu = "royalblue", reddish = "tomato"), alpha = FALSE)
col2hex(c(blu = "royalblue", reddish = "tomato"), alpha = TRUE)
```

---

rlnorm2 *Reparametrized lognormal distribution*

Description

A lognormal distribution parametrized as mean (ybar) and SDlog.

Usage

```
rlnorm2(n, mean = exp(0.5), sdlog = 1)
```

Arguments

n	number of random numbers desired.
mean	mean.
sdlog	log scale standard deviation.

Details

Log scale mean is `log(mean) - sdlog^2/2`.

Value

Vector of random numbers.

Author(s)

Peter Solymos

See Also

`link{rlnorm}`

Examples

```
summary(rlnorm2(10^6, 1.3, 0.5)) # mean ~ 1.3
exp(log(1.3) - 0.5^2/2) # ~ median
```

---

rmvn *Multivariate normal distribution*

Description

A shim of `mvrnorm` to return matrix when n < 2.

Usage

```
rmvn(n = 1L, mu, Sigma, ...)
```

Arguments

`n`	number of random vectors desired (nonnegative integer, can be 0).
`mu`	mean vector.
`Sigma`	variance-covariance matrix.
`...`	other arguments passed to `mvrnorm`.

Value

A matrix with `n` rows and `length(mu)` columns.

Author(s)

Peter Solymos

See Also

`mvrnorm`

Examples

```
rmvn(0, c(a=0, b=0), diag(1, 2, 2))
rmvn(1, c(a=0, b=0), diag(1, 2, 2))
rmvn(2, c(a=0, b=0), diag(1, 2, 2))

sapply(0:10, function(n) dim(rmvn(n, c(a=0, b=0), diag(1, 2, 2))))
```

---

run_app *Run Shiny apps*

Description

Run the Shiny apps that are included in the bSims package.

Usage

```
run_app(app = c("bsimsH", "bsimsHER", "distfunH", "distfunHER"))
```

Arguments

app	character, which app to run.

Details

"bsimsH": explore simulation settings in a single stratum.

"bsimsHER": explore simulation settings in multiple strata.

"distfunH": explore distance functions through a single stratum.

"distfunHER": explore distance functions through multiple strata with segmented sound attenuation (see dist_fun2).

See Also

bsims_init

Index