Package: nimbleSCR (via r-universe)

August 25, 2024

Type Package

Title Spatial Capture-Recapture (SCR) Methods Using 'nimble'

Version 0.2.1

Maintainer Daniel Turek <danielturek@gmail.com>

Date 2022-11-30

Description Provides utility functions, distributions, and fitting methods for Bayesian Spatial Capture-Recapture (SCR) and Open Population Spatial Capture-Recapture (OPSCR) modelling using the nimble package (de Valpine et al. 2017 <doi:10.1080/10618600.2016.1172487 >). Development of the package was motivated primarily by the need for flexible and efficient analysis of large-scale SCR data (Bischof et al. 2020 <doi:10.1073/pnas.2011383117 >). Computational methods and techniques implemented in nimbleSCR include those discussed in Turek et al. 2021 <doi:10.1002/ecs2.3385>; among others. For a recent application of nimbleSCR, see Milleret et al. (2021) <doi:10.1098/rsbl.2021.0128>.

License GPL-3

Depends R (>= 3.5.0), nimble

Imports methods

Encoding UTF-8

RoxygenNote 7.2.1

Suggests knitr, rmarkdown, testthat (>= 3.0.0), coda, basicMCMCplots

VignetteBuilder knitr

Config/testthat/edition 3

Collate calcWindowSizes.R getWindowIndex.R integrateIntensityLocal_normal.R integrateIntensityLocal_exp.R integrateIntensity_normal.R integrateIntensity_exp.R stratRejectionSampler_normal.R stratRejectionSampler_exp.R dDispersal_exp.R dHabitatMask.R dbernppAC.R dbernppACmovement_normal.R dbernppACmovement_exp.R dbernppDetection_normal.R dbernppLocalACmovement_normal.R

dbernppLocalACmovement_exp.R dbernppLocalDetection_normal.R
dbinomLocal_normal.R dmultiLocal_normal.R dbinom_vector.R
dnormalizer.R dpoisLocal_normal.R dbinomLocal_normalPlateau.R
dbinomLocal_exp.R dpoisppAC.R dpoisppDetection_normal.R
dpoisppLocalDetection_normal.R getLocalObjects.R
getMidPointNodes.R getWindowCoords.R getSparseY.R
getHomeRangeArea.R localTrapCalculations.R
makeConstantNimbleFunction.R marginalVoidProbIntegrand.R
marginalVoidProbNumIntegration.R scaleCoordsToHabitatGrid.R
dcatState1Alive1Dead.R dcatState1Alive2Dead.R
dcatState2Alive2Dead.R sampler_categorical_general.R
calculateDensity.R zzz.R

NeedsCompilation no

Author Richard Bischof [aut], Daniel Turek [aut, cre], Cyril Milleret [aut], Torbjørn Ergon [aut], Pierre Dupont [aut], Soumen Dey [aut], Wei Zhang [aut], Perry de Valpine [aut]

Date/Publication 2022-11-30 15:30:02 UTC

Repository https://danielturek.r-universe.dev

RemoteUrl https://github.com/cran/nimbleSCR

RemoteRef HEAD

RemoteSha 8449287983781e401dc883c7fd85e08f7f576025

Contents

`calculateDensity` *NIMBLE function to calculate the density of individuals alive in each habitat cell.*

Description

`calculateDensity` is a NIMBLE function to calculate number of individual activity centers (s) in each habitat cell.

Usage

```
calculateDensity(s, habitatGrid, indicator, numWindows, nIndividuals)
```

Arguments

`s`	`Matrix` of x- and y-coordinates of individual AC locations.
`habitatGrid`	Matrix of habitat window indices. Cell values should correspond to the order of habitat windows in spatial probabilities (e.g. `prob1To2Hab` as used in the function `dcatState1Alive2Dead`) or in `lowerCoords` and `upperCoords` as used in the `dbernppAC` function. #' @param indicator `Vector` of binary arguments specifying whether the individuals are considered alive (indicator = 1) or not (indicator = 0).
`indicator`	`Vector` of binary arguments specifying whether the individuals are considered alive (indicator = 1) or not (indicator = 0).
`numWindows`	`Scalar` Number of habitat windows.
`nIndividuals`	`Scalar` Number of individuals.

Author(s)

Cyril Milleret

Examples

```
lowerCoords <- matrix(c(0, 0, 1, 0, 0, 1, 1, 1), nrow = 4, byrow = TRUE)
upperCoords <- matrix(c(1, 1, 2, 1, 1, 2, 2, 2), nrow = 4, byrow = TRUE)
logIntensities <- log(rep(1,4))
logSumIntensity <- log(sum(c(1:4)))
habitatGrid <- matrix(c(1:4), nrow = 2, byrow = TRUE)
numGridRows <- nrow(habitatGrid)
numGridCols <- ncol(habitatGrid)

s <- matrix(NA,nrow=10,ncol=2)
for(i in 1:10){
  s[i,] <- rbernppAC(n=1, lowerCoords, upperCoords, logIntensities, logSumIntensity,
                     habitatGrid, numGridRows, numGridCols)
}

calculateDensity(s = s,
                 habitatGrid = habitatGrid,
                 indicator = rep(1, 10),
                 numWindows = prod(dim(habitatGrid)),
                 nIndividuals = 10
)
```

---

calcWindowSizes *Window size calculation*

Description

Calculates the sizes of a set of windows based on their lower and upper coordinates of each dimension. Can be applied to detection and habitat windows.

Usage

```
calcWindowSizes(lowerCoords, upperCoords)
```

Arguments

lowerCoords	Matrix of lower x- and y-coordinates of all windows. One row for each window.
upperCoords	Matrix of upper x- and y-coordinates of all windows. One row for each window.

Value

A vector of window sizes.

Author(s)

Wei Zhang

Examples

```
lowerCoords <- matrix(c(0, 0, 1, 0, 0, 1, 1, 1), nrow = 4, byrow = TRUE)
upperCoords <- matrix(c(1, 1, 3, 1, 1, 4, 3, 4), nrow = 4, byrow = TRUE)
calcWindowSizes(lowerCoords, upperCoords)
```

---

dbernppAC *Bernoulli point process for the distribution of activity centers*

Description

Density and random generation functions of the Bernoulli point process for the distribution of activity centers.

Usage

```
dbernppAC(
  x,
  lowerCoords,
  upperCoords,
  logIntensities,
  logSumIntensity,
  habitatGrid,
  numGridRows,
  numGridCols,
  log = 0
)

rbernppAC(
  n,
  lowerCoords,
  upperCoords,
  logIntensities,
  logSumIntensity,
  habitatGrid,
  numGridRows,
  numGridCols
)
```

Arguments

x	Vector of x- and y-coordinates of a single spatial point (i.e. AC location) scaled to the habitat (see (scaleCoordsToHabitatGrid).
lowerCoords, upperCoords	Matrices of lower and upper x- and y-coordinates of all habitat windows scaled to the habitat (see (scaleCoordsToHabitatGrid). One row for each window. Each window should be of size 1x1.
logIntensities	Vector of log habitat intensities for all habitat windows.
logSumIntensity	Log of the sum of habitat intensities over all windows.
habitatGrid	Matrix of habitat window indices. Cell values should correspond to the order of habitat windows in lowerCoords, upperCoords, and logIntensities. When the habitat grid only consists of a single row or column of windows, an additional row or column of dummy indices has to be added because the nimble model code requires a matrix.
numGridRows, numGridCols	Numbers of rows and columns of the habitat grid.
log	Logical argument, specifying whether to return the log-probability of the distribution.
n	Integer specifying the number of realisations to generate. Only n = 1 is supported.

Details

The dbernppAC distribution is a NIMBLE custom distribution which can be used to model and simulate the activity center location (x) of a single individual in continuous space over a set of habitat windows defined by their upper and lower coordinates (*lowerCoords,upperCoords*). The distribution assumes that the activity center follows a Bernoulli point process with intensity = *exp(logIntensities)*.

Value

dbernppAC gives the (log) probability density of the observation vector x. rbernppAC gives coordinates of a randomly generated spatial point.

Author(s)

Wei Zhang

References

W. Zhang, J. D. Chipperfield, J. B. Illian, P. Dupont, C. Milleret, P. de Valpine and R. Bischof. 2020.
A hierarchical point process model for spatial capture-recapture data. bioRxiv. DOI 10.1101/2020.10.06.325035

Examples

```
# Use the distribution in R
lowerCoords <- matrix(c(0, 0, 1, 0, 0, 1, 1, 1), nrow = 4, byrow = TRUE)
upperCoords <- matrix(c(1, 1, 2, 1, 1, 2, 2, 2), nrow = 4, byrow = TRUE)
logIntensities <- log(c(1:4))
logSumIntensity <- log(sum(c(1:4)))
habitatGrid <- matrix(c(1:4), nrow = 2, byrow = TRUE)
numGridRows <- nrow(habitatGrid)
numGridCols <- ncol(habitatGrid)
dbernppAC(c(0.5, 1.5), lowerCoords, upperCoords, logIntensities, logSumIntensity,
          habitatGrid, numGridRows, numGridCols, log = TRUE)
```

---

dbernppACmovement_exp *Bernoulli point process for activity center movement (exponential kernel)*

Description

Density and random generation functions of the Bernoulli point process for activity center movement between occasions based on a bivariate exponential distribution.

Usage

```
dbernppACmovement_exp(
  x,
  lowerCoords,
  upperCoords,
  s,
  lambda = -999,
  rate,
  baseIntensities,
  habitatGrid,
  numGridRows,
  numGridCols,
  numWindows,
  log = 0
)

rbernppACmovement_exp(
  n,
  lowerCoords,
  upperCoords,
  s,
  lambda = -999,
  rate,
  baseIntensities,
  habitatGrid,
```

```
  numGridRows,
  numGridCols,
  numWindows
)
```

Arguments

x	Vector of x- and y-coordinates of a single spatial point (typically AC location at time t+1) scaled to the habitat (see (scaleCoordsToHabitatGrid).
lowerCoords, upperCoords	Matrices of lower and upper x- and y-coordinates of all habitat windows. One row for each window. Each window should be of size 1x1 (after rescaling if necessary).
s	Vector of x- and y-coordinates of the isotropic multivariate exponential distribution mean (AC location at time t).
lambda	Rate parameter of the isotropic bivariate exponential distribution. Soon deprecated, use argument "rate" instead.
rate	Rate parameter of the isotropic multivariate exponential distribution.
baseIntensities	Vector of baseline habitat intensities for all habitat windows.
habitatGrid	Matrix of habitat window indices. Cell values should correspond to the order of habitat windows in lowerCoords and upperCoords. When the habitat grid only consists of a single row or column of windows, an additional row or column of dummy indices has to be added because the nimble model code requires a matrix.
numGridRows, numGridCols	Numbers of rows and columns of the habitat grid.
numWindows	Number of habitat windows. This value (positive integer) can be used to truncate lowerCoords and upperCoords so that extra rows beyond numWindows are ignored.
log	Logical argument, specifying whether to return the log-probability of the distribution.
n	Integer specifying the number of realisations to generate. Only n = 1 is supported.

Details

The dbernppACmovement_exp distribution is a NIMBLE custom distribution which can be used to model and simulate movement of activity centers between consecutive occasions in open population models. The distribution assumes that the new individual activity center location (x) follows an isotropic exponential normal centered on the previous activity center (s) with rate (*lambda*).

Value

dbernppACmovement_exp gives the (log) probability density of the observation vector x. rbernppACmovement_exp gives coordinates of a randomly generated spatial point.

Author(s)

Wei Zhang and Cyril Milleret

References

W. Zhang, J. D. Chipperfield, J. B. Illian, P. Dupont, C. Milleret, P. de Valpine and R. Bischof. 2020.
A hierarchical point process model for spatial capture-recapture data. bioRxiv. DOI 10.1101/2020.10.06.325035

Examples

```
# Use the distribution in R
lowerCoords <- matrix(c(0, 0, 1, 0, 0, 1, 1, 1), nrow = 4, byrow = TRUE)
upperCoords <- matrix(c(1, 1, 2, 1, 1, 2, 2, 2), nrow = 4, byrow = TRUE)
s <- c(1, 1) # Currrent activity center location
rate <- 0.1
baseIntensities <- c(1:4)
habitatGrid <- matrix(c(1:4), nrow = 2, byrow = TRUE)
numRows <- nrow(habitatGrid)
numCols <- ncol(habitatGrid)
numWindows <- 4
# The log probability density of moving from (1,1) to (1.2, 0.8)
dbernppACmovement_exp(x = c(1.2, 0.8),
lowerCoords = lowerCoords,
upperCoords = upperCoords,
s = s,
rate = rate,
baseIntensities = baseIntensities,
habitatGrid = habitatGrid,
numGridRows = numRows,
numGridCols = numCols,
numWindows = numWindows,
log = TRUE)
```

---

dbernppACmovement_normal

Bernoulli point process for activity center movement (normal kernel)

Description

Density and random generation functions of the Bernoulli point process for activity center movement between occasions based on a bivariate normal distribution.

Usage

```
dbernppACmovement_normal(
  x,
  lowerCoords,
  upperCoords,
  s,
```

```
  sd,
  baseIntensities,
  habitatGrid,
  numGridRows,
  numGridCols,
  numWindows,
  log = 0
)

rbernppACmovement_normal(
  n,
  lowerCoords,
  upperCoords,
  s,
  sd,
  baseIntensities,
  habitatGrid,
  numGridRows,
  numGridCols,
  numWindows
)
```

Arguments

`x`	Vector of x- and y-coordinates of a single spatial point (typically AC location at time t+1) scaled to the habitat (see (`scaleCoordsToHabitatGrid`).
`lowerCoords, upperCoords`	Matrices of lower and upper x- and y-coordinates of all habitat windows scaled to the habitat (see (`scaleCoordsToHabitatGrid`). One row for each window. Each window should be of size 1x1.
`s`	Vector of x- and y-coordinates of the isotropic bivariate normal distribution mean (AC location at time t).
`sd`	Standard deviation of the isotropic bivariate normal distribution..
`baseIntensities`	Vector of baseline habitat intensities for all habitat windows.
`habitatGrid`	Matrix of habitat window indices. Cell values should correspond to the order of habitat windows in `lowerCoords` and `upperCoords`. When the habitat grid only consists of a single row or column of windows, an additional row or column of dummy indices has to be added because the `nimble` model code requires a matrix.
`numGridRows, numGridCols`	Numbers of rows and columns of the habitat grid.
`numWindows`	Number of habitat windows. This value (positive integer) can be used to truncate `lowerCoords` and `upperCoords` so that extra rows beyond `numWindows` are ignored.
`log`	Logical argument, specifying whether to return the log-probability of the distribution.

n Integer specifying the number of realisations to generate. Only n = 1 is supported.

Details

The dbernppACmovement_normal distribution is a NIMBLE custom distribution which can be used to model and simulate movement of activity centers between consecutive occasions in open population models. The distribution assumes that the new individual activity center location (x) follows an isotropic multivariate normal centered on the previous activity center (s) with standard deviation (sd).

Value

dbernppACmovement_normal gives the (log) probability density of the observation vector x. rbernppACmovement_normal gives coordinates of a randomly generated spatial point.

Author(s)

Wei Zhang and Cyril Milleret

References

W. Zhang, J. D. Chipperfield, J. B. Illian, P. Dupont, C. Milleret, P. de Valpine and R. Bischof. 2020.
A hierarchical point process model for spatial capture-recapture data. bioRxiv. DOI 10.1101/2020.10.06.325035

Examples

```
# Use the distribution in R
lowerCoords <- matrix(c(0, 0, 1, 0, 0, 1, 1, 1), nrow = 4, byrow = TRUE)
upperCoords <- matrix(c(1, 1, 2, 1, 1, 2, 2, 2), nrow = 4, byrow = TRUE)
s <- c(1, 1) # Currrent activity center location
sd <- 0.1
baseIntensities <- c(1:4)
habitatGrid <- matrix(c(1:4), nrow = 2, byrow = TRUE)
numRows <- nrow(habitatGrid)
numCols <- ncol(habitatGrid)
numWindows <- 4
# The log probability density of moving from (1,1) to (1.2, 0.8)
dbernppACmovement_normal(c(1.2, 0.8), lowerCoords, upperCoords, s, sd, baseIntensities,
                         habitatGrid, numRows, numCols, numWindows, log = TRUE)
```

---

dbernppDetection_normal

Bernoulli point process detection model

Description

Density and random generation functions of the Bernoulli point process for detection.

Usage

```
dbernppDetection_normal(
  x,
  lowerCoords,
  upperCoords,
  s,
  sd,
  baseIntensities,
  numWindows,
  indicator,
  log = 0
)

rbernppDetection_normal(
  n,
  lowerCoords,
  upperCoords,
  s,
  sd,
  baseIntensities,
  numWindows,
  indicator
)
```

Arguments

x	Vector with three elements representing the x- and y-coordinates and the id of the corresponding detection window for a single spatial point (detection location) scaled to the habitat (see (scaleCoordsToHabitatGrid).
lowerCoords, upperCoords	Matrices of lower and upper x- and y-coordinates of all detection windows scaled to the habitat (see (scaleCoordsToHabitatGrid). One row for each window. Each window should be of size 1x1.
s	VVector of x- and y-coordinates of the isotropic bivariate normal distribution mean (i.e. the AC location)..
sd	Standard deviation of the isotropic bivariate normal distribution.
baseIntensities	Vector of baseline detection intensities for all detection windows.
numWindows	Number of detection windows. This value (positive integer) is used to truncate lowerCoords and upperCoords so that extra rows beyond numWindows are ignored.
indicator	Binary argument specifying whether the individual is available for detection (indicator = 1) or not (indicator = 0).
log	Logical argument, specifying whether to return the log-probability of the distribution.
n	Integer specifying the number of realisations to generate. Only n = 1 is supported.

Details

The dbernppDetection_normal distribution is a NIMBLE custom distribution which can be used to model and simulate Bernoulli observations (*x*) of a single individual in continuous space over a set of detection windows defined by their upper and lower coordinates (*lowerCoords,upperCoords*). The distribution assumes that an individual's detection probability follows an isotropic multivariate normal centered on the individual's activity center (*s*) with standard deviation (*sd*).

Value

dbernppDetection_normal gives the (log) probability density of the observation vector x. rbernppDetection_normal gives coordinates of a randomly generated spatial point.

Author(s)

Wei Zhang and Cyril Milleret

References

W. Zhang, J. D. Chipperfield, J. B. Illian, P. Dupont, C. Milleret, P. de Valpine and R. Bischof. 2020.
A hierarchical point process model for spatial capture-recapture data. bioRxiv. DOI 10.1101/2020.10.06.325035

Examples

```
coordsHabitatGridCenter <- matrix(c(0.5, 3.5,
                                    1.5, 3.5,
                                    2.5, 3.5,
                                    3.5, 3.5,
                                    0.5, 2.5,
                                    1.5, 2.5,
                                    2.5, 2.5,
                                    3.5, 2.5,
                                    0.5, 1.5,
                                    1.5, 1.5,
                                    2.5, 1.5,
                                    3.5, 1.5,
                                    0.5, 0.5,
                                    1.5, 0.5,
                                    2.5, 0.5,
                                    3.5, 0.5), ncol = 2,byrow = TRUE)
colnames(coordsHabitatGridCenter) <- c("x","y")
# Create observation windows
 lowerCoords <- matrix(c(0, 0, 1, 0, 0, 1, 1, 1), nrow = 4, byrow = TRUE)
 upperCoords <- matrix(c(1, 1, 2, 1, 1, 2, 2, 2), nrow = 4, byrow = TRUE)
 colnames(lowerCoords) <- colnames(upperCoords) <- c("x","y")
# Rescale coordinates
ScaledLowerCoords <- scaleCoordsToHabitatGrid(coordsData =  lowerCoords,
                                    coordsHabitatGridCenter = coordsHabitatGridCenter)
ScaledUpperCoords <- scaleCoordsToHabitatGrid(coordsData =  upperCoords,
                                    coordsHabitatGridCenter = coordsHabitatGridCenter)
ScaledUpperCoords$coordsDataScaled[,2] <- ScaledUpperCoords$coordsDataScaled[,2] + 1.5
ScaledLowerCoords$coordsDataScaled[,2] <- ScaledLowerCoords$coordsDataScaled[,2] - 1.5
```

```
s <- c(1, 1)
sd <- 0.1
baseIntensities <- c(1:4)
windowIndex <- 4
numPoints <- 1
numWindows <- 4
indicator <- 1
x <- c(0.5, 2)
windowIndex <- getWindowIndex(curCoords = x,
                              lowerCoords = ScaledLowerCoords$coordsDataScaled,
                              upperCoords =ScaledUpperCoords$coordsDataScaled)
x <- c(x, windowIndex)

dbernppDetection_normal(x, lowerCoords, upperCoords,
                        s, sd, baseIntensities
                        , numWindows,
                        indicator, log = TRUE)
```

---

dbernppLocalACmovement_exp

Local evaluation of a Bernoulli point process for activity center movement (exponential kernel)

Description

Density and random generation functions of the Bernoulli point process for activity center movement between occasions based on a bivariate exponential distribution and local evaluation.

Usage

```
dbernppLocalACmovement_exp(
  x,
  lowerCoords,
  upperCoords,
  s,
  lambda = -999,
  rate,
  baseIntensities,
  habitatGrid,
  habitatGridLocal,
  resizeFactor = 1,
  localHabWindowIndices,
  numLocalHabWindows,
  numGridRows,
  numGridCols,
  numWindows,
```

```
  log = 0
)

rbernppLocalACmovement_exp(
  n,
  lowerCoords,
  upperCoords,
  s,
  lambda = -999,
  rate,
  baseIntensities,
  habitatGrid,
  habitatGridLocal,
  resizeFactor = 1,
  localHabWindowIndices,
  numLocalHabWindows,
  numGridRows,
  numGridCols,
  numWindows
)
```

Arguments

x	Vector of x- and y-coordinates of a single spatial point (typically AC location at time t+1) scaled to the habitat (see (scaleCoordsToHabitatGrid).
lowerCoords, upperCoords	Matrices of lower and upper x- and y-coordinates of all habitat windows. One row for each window. Each window should be of size 1x1 (after rescaling if necessary).
s	Vector of x- and y-coordinates of the isotropic multivariate exponential distribution mean (AC location at time t).
lambda	Rate parameter of the isotropic bivariate exponential distribution. Soon deprecated, use argument "rate" instead.
rate	Rate parameter of the isotropic bivariate exponential distribution.
baseIntensities	Vector of baseline habitat intensities for all habitat windows.
habitatGrid	Matrix of habitat window indices. Cell values should correspond to the order of habitat windows in lowerCoords and upperCoords. When the habitat grid only consists of a single row or column of windows, an additional row or column of dummy indices has to be added because the nimble model code requires a matrix.
habitatGridLocal	Matrix of rescaled habitat grid cells indices, as returned by the getLocalObjects function (object named habitatGrid).
resizeFactor	Aggregation factor used in the getLocalObjects function to reduce the number of habitat grid cells.

`localHabWindowIndices`
: Matrix of indices of local habitat windows around each local habitat grid cell (`habitatGridLocal`) from localIndices returned by `getLocalObjects` function.

`numLocalHabWindows`
: Vector of numbers of local habitat windows around all habitat grid cells, from `numLocalIndices` returned by the `getLocalObjects` function. The ith number gives the number of local (original) habitat windows for the ith local habitat grid cell `habitatGridLocal`.

`numGridRows`, `numGridCols`
: Numbers of rows and columns of the `habitatGrid`.

`numWindows`
: Number of habitat windows. This value (positive integer) is used to truncate `lowerCoords` and `upperCoords` so that extra rows beyond `numWindows` are ignored.

`log`
: Logical argument, specifying whether to return the log-probability of the distribution.

`n`
: Integer specifying the number of realisations to generate. Only n = 1 is supported.

Details

The `dbernppLocalACmovement_exp` distribution is a NIMBLE custom distribution which can be used to model and simulate movement of activity centers between consecutive occasions in open population models. The distribution assumes that the new individual activity center location (x) follows an isotropic exponential normal centered on the previous activity center (s) with rate ($lambda$). The local evaluation approach is implemented.

Value

The (log) probability density of the observation vector `x`.

Author(s)

Wei Zhang and Cyril Milleret

References

W. Zhang, J. D. Chipperfield, J. B. Illian, P. Dupont, C. Milleret, P. de Valpine and R. Bischof. 2020. A hierarchical point process model for spatial capture-recapture data. bioRxiv. DOI 10.1101/2020.10.06.325035

C. Milleret, P. Dupont, C. Bonenfant, H. Broseth, O. Flagstad, C. Sutherland and R. Bischof. 2019. A local evaluation of the individual state-space to scale up Bayesian spatial capture-recapture. Ecology and Evolution 9:352-363

Examples

```
# Creat habitat grid
habitatGrid <- matrix(c(1:(4^2)), nrow = 4, ncol=4, byrow = TRUE)
coordsHabitatGridCenter <- matrix(c(0.5, 3.5,
                                    1.5, 3.5,
```

```
                                    2.5, 3.5,
                                    3.5, 3.5,
                                    0.5, 2.5,
                                    1.5, 2.5,
                                    2.5, 2.5,
                                    3.5, 2.5,
                                    0.5, 1.5,
                                    1.5, 1.5,
                                    2.5, 1.5,
                                    3.5, 1.5,
                                    0.5, 0.5,
                                    1.5, 0.5,
                                    2.5, 0.5,
                                    3.5, 0.5), ncol = 2,byrow = TRUE)
colnames(coordsHabitatGridCenter) <- c("x","y")
# Create habitat windows
lowerCoords <- coordsHabitatGridCenter-0.5
upperCoords <- coordsHabitatGridCenter+0.5
colnames(lowerCoords) <- colnames(upperCoords) <- c("x","y")
# Plot check
plot(lowerCoords[,"y"]~lowerCoords[,"x"],pch=16, xlim=c(0,4), ylim=c(0,4),col="red")
points(upperCoords[,"y"]~upperCoords[,"x"],col="red",pch=16)
points(coordsHabitatGridCenter[,"y"]~coordsHabitatGridCenter[,"x"],pch=16)

# Rescale coordinates
ScaledLowerCoords <- scaleCoordsToHabitatGrid(coordsData =  lowerCoords,
                                      coordsHabitatGridCenter = coordsHabitatGridCenter)
ScaledUpperCoords <- scaleCoordsToHabitatGrid(coordsData =  upperCoords,
                                      coordsHabitatGridCenter = coordsHabitatGridCenter)
ScaledUpperCoords$coordsDataScaled[,2] <- ScaledUpperCoords$coordsDataScaled[,2] + 1
ScaledLowerCoords$coordsDataScaled[,2] <- ScaledLowerCoords$coordsDataScaled[,2] - 1
habitatMask <- matrix(1, nrow = 4, ncol=4, byrow = TRUE)
# Create local objects
HabWindowsLocal <- getLocalObjects(habitatMask = habitatMask,
                                   coords = coordsHabitatGridCenter,
                                   dmax=4,
                                   resizeFactor = 1,
                                   plot.check = TRUE
)

s <- c(1, 1) # Currrent activity center location
rate <- 0.1
numWindows <- nrow(coordsHabitatGridCenter)
baseIntensities <- rep(1,numWindows)
numRows <- nrow(habitatGrid)
numCols <- ncol(habitatGrid)

# The log probability density of moving from (1,1) to (1.2, 0.8)
dbernppLocalACmovement_exp(x = c(1.2, 0.8),
 lowerCoords =lowerCoords,
 upperCoords = upperCoords,
 s =s,
 rate = rate,
```

```
 baseIntensities = baseIntensities,
 habitatGrid = habitatGrid,
 habitatGridLocal = HabWindowsLocal$habitatGrid,
 resizeFactor = HabWindowsLocal$resizeFactor,
 localHabWindowIndices = HabWindowsLocal$localIndices,
 numLocalHabWindows = HabWindowsLocal$numLocalIndices,
 numGridRows = numRows,
 numGridCols = numCols,
 numWindows = numWindows,
 log = TRUE)
```

`dbernppLocalACmovement_normal` *Local evaluation of a Bernoulli point process for activity center movement (normal kernel)*

Description

Density and random generation functions of the Bernoulli point process for activity center movement between occasions based on a bivariate normal distribution and local evaluation.

Usage

```
dbernppLocalACmovement_normal(
  x,
  lowerCoords,
  upperCoords,
  s,
  sd,
  baseIntensities,
  habitatGrid,
  habitatGridLocal,
  resizeFactor = 1,
  localHabWindowIndices,
  numLocalHabWindows,
  numGridRows,
  numGridCols,
  numWindows,
  log = 0
)

rbernppLocalACmovement_normal(
  n,
  lowerCoords,
  upperCoords,
```

```
  s,
  sd,
  baseIntensities,
  habitatGrid,
  habitatGridLocal,
  resizeFactor = 1,
  localHabWindowIndices,
  numLocalHabWindows,
  numGridRows,
  numGridCols,
  numWindows
)
```

Arguments

`x`	Vector of x- and y-coordinates of a single spatial point (typically AC location at time t+1) scaled to the habitat (see (`scaleCoordsToHabitatGrid`).
`lowerCoords, upperCoords`	Matrices of lower and upper x- and y-coordinates of all habitat windows. One row for each window. Each window should be of size 1x1 (after rescaling if necessary).
`s`	Vector of x- and y-coordinates of the isotropic bivariate normal distribution mean (i.e. the AC location).
`sd`	Standard deviation of the isotropic bivariate normal distribution.
`baseIntensities`	Vector of baseline habitat intensities for all habitat windows.
`habitatGrid`	Matrix of habitat window indices. Cell values should correspond to the order of habitat windows in `lowerCoords` and `upperCoords`. When the habitat grid only consists of a single row or column of windows, an additional row or column of dummy indices has to be added because the `nimble` model code requires a matrix.
`habitatGridLocal`	Matrix of rescaled habitat grid cells indices, from localIndices returned by the `getLocalObjects` function (
`resizeFactor`	Aggregation factor used in the `getLocalObjects` function to reduce the number of habitat grid cells.
`localHabWindowIndices`	Matrix of indices of local habitat windows around each local habitat grid cell (`habitatGridLocal`), from localIndices returned by the `getLocalObjects` function.
`numLocalHabWindows`	Vector of numbers of local habitat windows around all habitat grid cells, as returned by the getLocalObjects function (object named `numLocalIndices`). The ith number gives the number of local (original) habitat windows for the ith (rescaled) habitat window.
`numGridRows, numGridCols`	Numbers of rows and columns of the `habitatGrid`.

numWindows	Number of habitat windows. This value (positive integer) is used to truncate lowerCoords and upperCoords so that extra rows beyond numWindows are ignored.
log	Logical argument, specifying whether to return the log-probability of the distribution.
n	Integer specifying the number of realisations to generate. Only n = 1 is supported.

Details

The dbernppLocalACmovement_normal distribution is a NIMBLE custom distribution which can be used to model and simulate movement of activity centers between consecutive occasions in open population models. The distribution assumes that the new individual activity center location (x) follows an isotropic multivariate normal centered on the previous activity center (s) with standard deviation (sd). The local evaluation technique is implemented.

Value

The (log) probability density of the observation vector x.

Author(s)

Wei Zhang and Cyril Milleret

References

W. Zhang, J. D. Chipperfield, J. B. Illian, P. Dupont, C. Milleret, P. de Valpine and R. Bischof. 2020. A hierarchical point process model for spatial capture-recapture data. bioRxiv. DOI 10.1101/2020.10.06.325035

C. Milleret, P. Dupont, C. Bonenfant, H. Brøseth, Ø. Flagstad, C. Sutherland and R. Bischof. 2019. A local evaluation of the individual state-space to scale up Bayesian spatial capture-recapture. Ecology and Evolution 9:352-363

Examples

```
# Creat habitat grid
habitatGrid <- matrix(c(1:(4^2)), nrow = 4, ncol=4, byrow = TRUE)
coordsHabitatGridCenter <- matrix(c(0.5, 3.5,
                                    1.5, 3.5,
                                    2.5, 3.5,
                                    3.5, 3.5,
                                    0.5, 2.5,
                                    1.5, 2.5,
                                    2.5, 2.5,
                                    3.5, 2.5,
                                    0.5, 1.5,
                                    1.5, 1.5,
                                    2.5, 1.5,
                                    3.5, 1.5,
                                    0.5, 0.5,
                                    1.5, 0.5,
```

```
                                    2.5, 0.5,
                                    3.5, 0.5), ncol = 2,byrow = TRUE)
colnames(coordsHabitatGridCenter) <- c("x","y")
# Create habitat windows
lowerCoords <- coordsHabitatGridCenter-0.5
upperCoords <- coordsHabitatGridCenter+0.5
colnames(lowerCoords) <- colnames(upperCoords) <- c("x","y")
# Plot check
plot(lowerCoords[,"y"]~lowerCoords[,"x"],pch=16, xlim=c(0,4), ylim=c(0,4),col="red")
points(upperCoords[,"y"]~upperCoords[,"x"],col="red",pch=16)
points(coordsHabitatGridCenter[,"y"]~coordsHabitatGridCenter[,"x"],pch=16)

# Rescale coordinates
ScaledLowerCoords <- scaleCoordsToHabitatGrid(coordsData =  lowerCoords,
                                      coordsHabitatGridCenter = coordsHabitatGridCenter)
ScaledUpperCoords <- scaleCoordsToHabitatGrid(coordsData =  upperCoords,
                                      coordsHabitatGridCenter = coordsHabitatGridCenter)
ScaledUpperCoords$coordsDataScaled[,2] <- ScaledUpperCoords$coordsDataScaled[,2] + 1
ScaledLowerCoords$coordsDataScaled[,2] <- ScaledLowerCoords$coordsDataScaled[,2] - 1
habitatMask <- matrix(1, nrow = 4, ncol=4, byrow = TRUE)
# Create local objects
HabWindowsLocal <- getLocalObjects(habitatMask = habitatMask,
                                   coords = coordsHabitatGridCenter,
                                   dmax=4,
                                   resizeFactor = 1,
                                   plot.check = TRUE
)

s <- c(1, 1) # Currrent activity center location
sd <- 0.1
numWindows <- nrow(coordsHabitatGridCenter)
baseIntensities <- rep(1,numWindows)
numRows <- nrow(habitatGrid)
numCols <- ncol(habitatGrid)

# The log probability density of moving from (1,1) to (1.2, 0.8)
dbernppLocalACmovement_normal(x = c(1.2, 0.8), lowerCoords, upperCoords, s,
                              sd, baseIntensities, habitatGrid,
                              HabWindowsLocal$habitatGrid, HabWindowsLocal$resizeFactor,
                          HabWindowsLocal$localIndices, HabWindowsLocal$numLocalIndices,
                              numRows, numCols, numWindows, log = TRUE)
```

---

dbernppLocalDetection_normal

Local evaluation for a Bernoulli point process detection model

---

Description

Density and random generation functions of the Bernoulli point process for detection based on a bivariate normal distribution.

Usage

```
dbernppLocalDetection_normal(
  x,
  lowerCoords,
  upperCoords,
  s,
  sd,
  baseIntensities,
  habitatGridLocal,
  resizeFactor = 1,
  localObsWindowIndices,
  numLocalObsWindows,
  numWindows,
  indicator,
  log = 0
)

rbernppLocalDetection_normal(
  n,
  lowerCoords,
  upperCoords,
  s,
  sd,
  baseIntensities,
  habitatGridLocal,
  resizeFactor = 1,
  localObsWindowIndices,
  numLocalObsWindows,
  numWindows,
  indicator
)
```

Arguments

`x`	Vector with three elements representing the x- and y-coordinates (x[1:2]), and the corresponding id the detection window (x[3]) of a single spatial point (detection location) scaled to the habitat (see `scaleCoordsToHabitatGrid`).
`lowerCoords, upperCoords`	Matrices of lower and upper x- and y-coordinates of all detection windows. One row for each window. Each window should be of size 1x1 (after rescaling if necessary).
`s`	Vector of x- and y-coordinates of the isotropic bivariate normal distribution mean (i.e. the AC location).

`sd` Standard deviation of the bivariate normal distribution.

`baseIntensities`
Vector of baseline detection intensities for all detection windows.

`habitatGridLocal`
Matrix of rescaled habitat grid cells indices, as returned by the `getLocalObjects` function (object named `habitatGrid`).

`resizeFactor` Aggregation factor used in the `getLocalObjects` function to reduce the number of habitat grid cells.

`localObsWindowIndices`
Matrix of indices of local observation windows around each local habitat grid cell (habitatGridLocal), from localIndices returned by the `getLocalObjects` function.

`numLocalObsWindows`
Vector of numbers of local observation windows around all habitat grid cells, as returned by the getLocalObjects function (object named `numLocalIndices`). The ith number gives the number of local (original) observation windows for the ith (rescaled) habitat window.

`numWindows` Number of detection windows. This value (positive integer) is used to truncate `lowerCoords` and `upperCoords` so that extra rows beyond `numWindows` are ignored.

`indicator` Binary argument specifying whether the individual is available for detection (indicator = 1) or not (indicator = 0).

`log` Logical argument, specifying whether to return the log-probability of the distribution.

`n` Integer specifying the number of realizations to generate. Only n = 1 is supported.

Details

The `dbernppDetection_normal` distribution is a NIMBLE custom distribution which can be used to model and simulate Bernoulli observations (x) of a single individual in continuous space over a set of detection windows defined by their upper and lower coordinates (*lowerCoords,upperCoords*). The distribution assumes that an individual's detection probability follows an isotropic multivariate normal centered on the individual's activity center (s) with standard deviation (*sd*). The local evaluation approach is implemented.

Value

The (log) probability density of the observation vector `x`.

Author(s)

Wei Zhang and Cyril Milleret

References

W. Zhang, J. D. Chipperfield, J. B. Illian, P. Dupont, C. Milleret, P. de Valpine and R. Bischof. 2020. A hierarchical point process model for spatial capture-recapture data. bioRxiv. DOI 10.1101/2020.10.06.325035

C. Milleret, P. Dupont, C. Bonenfant, H. Brøseth, Ø. Flagstad, C. Sutherland and R. Bischof. 2019. A local evaluation of the individual state-space to scale up Bayesian spatial capture-recapture. Ecology and Evolution 9:352-363

Examples

```
# Create habitat grid
coordsHabitatGridCenter <- matrix(c(0.5, 3.5,
                                    1.5, 3.5,
                                    2.5, 3.5,
                                    3.5, 3.5,
                                    0.5, 2.5,
                                    1.5, 2.5,
                                    2.5, 2.5,
                                    3.5, 2.5,
                                    0.5, 1.5,
                                    1.5, 1.5,
                                    2.5, 1.5,
                                    3.5, 1.5,
                                    0.5, 0.5,
                                    1.5, 0.5,
                                    2.5, 0.5,
                                    3.5, 0.5), ncol = 2,byrow = TRUE)
colnames(coordsHabitatGridCenter) <- c("x","y")
# Create observation windows
lowerCoords <- matrix(c(1, 1, 2, 1, 1, 2, 2, 2), nrow = 4, byrow = TRUE)
upperCoords <- matrix(c(2, 2, 3, 2, 2, 3, 3, 3), nrow = 4, byrow = TRUE)
colnames(lowerCoords) <- colnames(upperCoords) <- c("x","y")
# Plot check
plot(coordsHabitatGridCenter[,"y"]~coordsHabitatGridCenter[,"x"],pch=16)
points(lowerCoords[,"y"]~lowerCoords[,"x"],col="red",pch=16)
points(upperCoords[,"y"]~upperCoords[,"x"],col="red",pch=16)
#'
s <- c(1, 1)
sd <- 0.1
baseIntensities <- c(1:4)
windowIndex <- 4
numPoints <- 1
numWindows <- 4
indicator <- 1

# Rescale coordinates
ScaledLowerCoords <- scaleCoordsToHabitatGrid(coordsData =  lowerCoords,
                                    coordsHabitatGridCenter = coordsHabitatGridCenter)
ScaledUpperCoords <- scaleCoordsToHabitatGrid(coordsData =  upperCoords,
                                    coordsHabitatGridCenter = coordsHabitatGridCenter)
ScaledUpperCoords$coordsDataScaled[,2] <- ScaledUpperCoords$coordsDataScaled[,2] + 1.5
ScaledLowerCoords$coordsDataScaled[,2] <- ScaledLowerCoords$coordsDataScaled[,2] - 1.5
```

```
habitatMask <- matrix(1, nrow = 4, ncol=4, byrow = TRUE)
# Create local objects
ObsWindowsLocal <- getLocalObjects(habitatMask = habitatMask,
                                   coords = ScaledLowerCoords$coordsDataScaled,
                                   dmax=3,
                                   resizeFactor = 1,
                                   plot.check = TRUE
)
x <- c(1.1, 1.2)
windowIndex <- getWindowIndex(curCoords = x,
                              lowerCoords = ScaledLowerCoords$coordsDataScaled,
                              upperCoords =ScaledUpperCoords$coordsDataScaled)
x <- c(x, windowIndex)
dbernppLocalDetection_normal(x, ScaledLowerCoords$coordsDataScaled,
                             ScaledUpperCoords$coordsDataScaled,
                             s, sd, baseIntensities,
                             ObsWindowsLocal$habitatGrid, ObsWindowsLocal$resizeFactor,
                           ObsWindowsLocal$localIndices,ObsWindowsLocal$numLocalIndices,
                             numWindows, indicator, log = TRUE)
```

`dbinomLocal_exp` *Local evaluation of a binomial SCR observation process*

Description

The `dbinomLocal_exp` distribution is a NIMBLE custom distribution which can be used to model and simulate binomial observations (x) of a single individual over a set of traps defined by their coordinates *trapCoords* the distribution assumes that an individual's detection probability at any trap follows an exponential function of the distance between the individual's activity center (s) and the trap location. All coordinates (`s` and `trapCoords`) should be scaled to the habitat (see `scaleCoordsToHabitatGrid`)

Usage

```
dbinomLocal_exp(
  x,
  detNums = -999,
  detIndices,
  size,
  p0 = -999,
  p0Traps,
  rate,
  s,
  trapCoords,
  localTrapsIndices,
  localTrapsNum,
  resizeFactor = 1,
  habitatGrid,
```

```
  indicator,
  lengthYCombined = 0,
  log = 0
)

rbinomLocal_exp(
  n = 1,
  detNums = -999,
  detIndices,
  size,
  p0 = -999,
  p0Traps,
  rate,
  s,
  trapCoords,
  localTrapsIndices,
  localTrapsNum,
  resizeFactor = 1,
  habitatGrid,
  indicator,
  lengthYCombined = 0
)
```

Arguments

x	Vector of individual detection frequencies. This argument can be provided in two formats: (i) with the *y* object as returned by getSparseY; (ii) with the *yCombined* object as returned by getSparseY Note that when the random generation functionality is used (rbinomLocal_normal), only the yCombined format can be used. The *yCombined* object combines *detNums*, *x*, and *detIndices* (in that order). When such consolidated representation of the detection data x is used, *detIndices* and *detNums* arguments should not be specified.
detNums	Number of traps with at least one detection recorded in *x*; from the *detNums* object returned by the getSparseY function. This argument should not be specified when the *yCombined* object (returned by getSparseY) is provided as *x* and when detection data are simulated.
detIndices	Vector of indices of traps where the detections in *x* were recorded; from the *detIndices* object returned by the getSparseY function. This argument should not be specified when x is provided as the *yCombined* object (returned by getSparseY) and when detection data are simulated.
size	Vector of the number of trials (zero or more) for each trap (*trapCoords*).
p0	Baseline detection probability (scalar) used in the half-normal detection function. For trap-specific baseline detection probabilities use argument *p0Traps* (vector) instead.
p0Traps	Vector of baseline detection probabilities for each trap used in the half-normal detection function. When *p0Traps* is used, *p0* should not be provided.
rate	Rate parameter of the exponential detection function.

s	Individual activity center x- and y-coordinates scaled to the habitat (see (scaleCoordsToHabitatGrid).
trapCoords	Matrix of x- and y-coordinates of all traps scaled to the habitat (see (scaleCoordsToHabitatGrid).
localTrapsIndices	Matrix of indices of local traps around each habitat grid cell, as returned by the getLocalObjects function.
localTrapsNum	Vector of numbers of local traps around all habitat grid cells, as returned by the getLocalObjects function.
resizeFactor	Aggregation factor used in the getLocalObjects function to reduce the number of habitat grid cells to retrieve local traps for.
habitatGrid	Matrix of local habitat grid cell indices, from *habitatGrid* returned by the getLocalObjects function.
indicator	Binary argument specifying whether the individual is available for detection (indicator = 1) or not (indicator = 0).
lengthYCombined	The length of the x argument when the (*yCombined*) format of the detection data is provided; from the *lengthYCombined* object returned by getSparseY
log	Logical argument, specifying whether to return the log-probability of the distribution.
n	Integer specifying the number of realizations to generate. Only n = 1 is supported.

Details

The dbinomLocal_exp distribution incorporates three features to increase computation efficiency (see Turek et al., 2021 <doi.org/10.1002/ecs2.3385> for more details):

1. A local evaluation of the detection probability calculation (see Milleret et al., 2019 <doi:10.1002/ece3.4751> for more details)
2. A sparse matrix representation (*x*, *detIndices* and *detNums*) of the observation data to reduce the size of objects to be processed.
3. An indicator (*indicator*) to shortcut calculations for individuals unavailable for detection.

The dbinomLocal_exp distribution requires x- and y- detector coordinates (*trapCoords*) to be scaled to the habitat grid (*habitatGrid*) using the (scaleCoordsToHabitatGrid function.)

When the aim is to simulate detection data:

1. *x* should be provided using the *yCombined* object as returned by getSparseY,
2. arguments *detIndices* and *detNums* should not be provided,
3. argument *lengthYCombined* should be provided using the *lengthYCombined* object as returned by getSparseY.

Value

The log-likelihood value associated with the vector of detections, given the location of the activity center (s), and the exponential detection function : $p = p0 * exp(-rate * d)$.

Author(s)

Soumen Dey

References

Dey, S., Bischof, R., Dupont, P. P. A., & Milleret, C. (2022). Does the punishment fit the crime? Consequences and diagnosis of misspecified detection functions in Bayesian spatial capture–recapture modeling. Ecology and Evolution, 12, e8600. https://doi.org/10.1002/ece3.8600

Examples

```
# I. DATA SET UP
coordsHabitatGridCenter <- matrix(c(0.5, 3.5,
                                    1.5, 3.5,
                                    2.5, 3.5,
                                    3.5, 3.5,
                                    0.5, 2.5,
                                    1.5, 2.5,
                                    2.5, 2.5,
                                    3.5, 2.5,
                                    0.5, 1.5,
                                    1.5, 1.5,
                                    2.5, 1.5,
                                    3.5, 1.5,
                                    0.5, 0.5,
                                    1.5, 0.5,
                                    2.5, 0.5,
                                    3.5, 0.5), ncol=2,byrow = TRUE)
colnames(coordsHabitatGridCenter) <- c("x","y")
# CREATE OBSERVATION WINDOWS
trapCoords <- matrix(c(1.5, 1.5, 2.5, 1.5, 1.5, 2.5, 2.5, 2.5), nrow = 4, byrow = TRUE)
colnames(trapCoords) <- c("x","y")
# PLOT CHECK
plot(coordsHabitatGridCenter[,"y"]~coordsHabitatGridCenter[,"x"],pch=16)
points(trapCoords[,"y"]~trapCoords[,"x"],col="red",pch=16)

# PARAMETERS
p0 <- 0.3
rate <- 1/1.5
indicator <- 1
# WE CONSIDER 2 INDIVIDUALS
y <- matrix(c(0, 1, 1, 0,
              0, 1, 0, 1),ncol=4,nrow=2)
s <- matrix(c(0.5, 1,
              1.6, 2.3),ncol=2,nrow=2)

# RESCALE COORDINATES
ScaledtrapCoords <- scaleCoordsToHabitatGrid(coordsData =  trapCoords,
                                     coordsHabitatGridCenter = coordsHabitatGridCenter)
ScaledtrapCoords<- ScaledtrapCoords$coordsDataScaled
habitatMask <- matrix(1, nrow = 4, ncol=4, byrow = TRUE)
```

```
# CREATE LOCAL OBJECTS
TrapLocal <- getLocalObjects(habitatMask = habitatMask,
                                  coords = ScaledtrapCoords,
                                  dmax=2.5,
                                  resizeFactor = 1,
                                  plot.check = TRUE
)

# GET SPARSE MATRIX
SparseY <- getSparseY(y)

# II. USING THE DENSITY FUNCTION
 # WE TAKE THE FIRST INDIVIDUAL
i=1
  # OPTION 1: USING THE RANDOM GENERATION FUNCTIONNALITY
dbinomLocal_exp(x=SparseY$y[i,,1],
                  detNums=SparseY$detNums[i],
                  detIndices=SparseY$detIndices[i,,1],
                  size=rep(1,4),
                  p0 = p0,
                  rate= rate,
                  s=s[i,1:2],
                  trapCoords=ScaledtrapCoords,
                  localTrapsIndices=TrapLocal$localIndices,
                  localTrapsNum=TrapLocal$numLocalIndices,
                  resizeFactor=TrapLocal$resizeFactor,
                  habitatGrid=TrapLocal$habitatGrid,
                  indicator=indicator)

  # OPTION 2: USING RANDOM GENERATION FUNCTIONNALITY
  # WE DO NOT PROVIDE THE detNums AND detIndices ARGUMENTS
dbinomLocal_exp(x=SparseY$yCombined[i,,1],
                  size=rep(1,4),
                  p0 = p0,
                  rate= rate,
                  s=s[i,1:2],
                  trapCoords=ScaledtrapCoords,
                  localTrapsIndices=TrapLocal$localIndices,
                  localTrapsNum=TrapLocal$numLocalIndices,
                  resizeFactor=TrapLocal$resizeFactor,
                  habitatGrid=TrapLocal$habitatGrid,
                  indicator=indicator,
                  lengthYCombined = SparseY$lengthYCombined)

# III. USING THE RANDOM GENERATION FUNCTION
rbinomLocal_exp(n=1,
                  size=rep(1,4),
                  p0 = p0,
                  rate= rate,
                  s=s[i,1:2],
                  trapCoords=ScaledtrapCoords,
                  localTrapsIndices=TrapLocal$localIndices,
```

```
                    localTrapsNum=TrapLocal$numLocalIndices,
                    resizeFactor=TrapLocal$resizeFactor,
                    habitatGrid=TrapLocal$habitatGrid,
                    indicator=indicator,
                    lengthYCombined = SparseY$lengthYCombined)
```

`dbinomLocal_normal` *Local evaluation of a binomial SCR detection process*

Description

The `dbinomLocal_normal` distribution is a NIMBLE custom distribution which can be used to model and simulate binomial observations (x) of a single individual over a set of traps defined by their coordinates *trapCoords* the distribution assumes that an individual's detection probability at any trap follows a half-normal function of the distance between the individual's activity center (s) and the trap location. All coordinates (`s` and `trapCoords`) should be scaled to the habitat (see `scaleCoordsToHabitatGrid`)

Usage

```
dbinomLocal_normal(
  x,
  detNums = -999,
  detIndices,
  size,
  p0 = -999,
  p0Traps,
  sigma,
  s,
  trapCoords,
  localTrapsIndices,
  localTrapsNum,
  resizeFactor = 1,
  habitatGrid,
  indicator,
  lengthYCombined = 0,
  log = 0
)

rbinomLocal_normal(
  n = 1,
  detNums = -999,
  detIndices,
  size,
  p0 = -999,
  p0Traps,
```

```
  sigma,
  s,
  trapCoords,
  localTrapsIndices,
  localTrapsNum,
  resizeFactor = 1,
  habitatGrid,
  indicator,
  lengthYCombined = 0
)
```

Arguments

x	Vector of individual detection frequencies. This argument can be provided in two formats: (i) with the *y* object as returned by getSparseY; (ii) with the *yCombined* object as returned by getSparseY Note that when the random generation functionality is used (rbinomLocal_normal), only the yCombined format can be used. The *yCombined* object combines *detNums*, *x*, and *detIndices* (in that order). When such consolidated representation of the detection data x is used, *detIndices* and *detNums* arguments should not be specified.
detNums	Number of traps with at least one detection recorded in *x*; from the *detNums* object returned by the getSparseY function. This argument should not be specified when the *yCombined* object (returned by getSparseY) is provided as *x* and when detection data are simulated.
detIndices	Vector of indices of traps where the detections in *x* were recorded; from the *detIndices* object returned by the getSparseY function. This argument should not be specified when x is provided as the *yCombined* object (returned by getSparseY) and when detection data are simulated.
size	Vector of the number of trials (zero or more) for each trap (*trapCoords*).
p0	Baseline detection probability (scalar) used in the half-normal detection function. For trap-specific baseline detection probabilities use argument *p0Traps* (vector) instead.
p0Traps	Vector of baseline detection probabilities for each trap used in the half-normal detection function. When *p0Traps* is used, *p0* should not be provided.
sigma	Scale parameter of the half-normal detection function.
s	Individual activity center x- and y-coordinates scaled to the habitat (see (scaleCoordsToHabitatGrid).
trapCoords	Matrix of x- and y-coordinates of all traps scaled to the habitat (see (scaleCoordsToHabitatGrid).
localTrapsIndices	Matrix of indices of local traps around each habitat grid cell, as returned by the getLocalObjects function.
localTrapsNum	Vector of numbers of local traps around all habitat grid cells, as returned by the getLocalObjects function.
resizeFactor	Aggregation factor used in the getLocalObjects function to reduce the number of habitat grid cells to retrieve local traps for.
habitatGrid	Matrix of local habitat grid cell indices, from *habitatGrid* returned by the getLocalObjects function.

`indicator`	Binary argument specifying whether the individual is available for detection (indicator = 1) or not (indicator = 0).
`lengthYCombined`	The length of the x argument when the (*yCombined*) format of the detection data is provided; from the *lengthYCombined* object returned by `getSparseY`
`log`	Logical argument, specifying whether to return the log-probability of the distribution.
`n`	Integer specifying the number of realizations to generate. Only n = 1 is supported.

Details

The `dbinomLocal_normal` distribution incorporates three features to increase computation efficiency (see Turek et al., 2021 <doi.org/10.1002/ecs2.3385> for more details):

1. A local evaluation of the detection probability calculation (see Milleret et al., 2019 <doi:10.1002/ece3.4751> for more details)
2. A sparse matrix representation (*x*, *detIndices* and *detNums*) of the observation data to reduce the size of objects to be processed.
3. An indicator (*indicator*) to shortcut calculations for individuals unavailable for detection.

The `dbinomLocal_normal` distribution requires x- and y- detector coordinates (*trapCoords*) and activity centers coordinates (*s*) to be scaled to the habitat grid (*habitatGrid*) using the (`scaleCoordsToHabitatGrid` function.)

When the aim is to simulate detection data:

1. *x* should be provided using the *yCombined* object as returned by `getSparseY`,
2. arguments *detIndices* and *detNums* should not be provided,
3. argument *lengthYCombined* should be provided using the *lengthYCombined* object as returned by `getSparseY`.

Value

The log-likelihood value associated with the vector of detections, given the location of the activity center (s), and the half-normal detection function : $p = p0 * exp(-d^2/2\sigma^2)$.

Author(s)

Cyril Milleret, Soumen Dey

Examples

```
# I. DATA SET UP
coordsHabitatGridCenter <- matrix(c(0.5, 3.5,
                                    1.5, 3.5,
                                    2.5, 3.5,
                                    3.5, 3.5,
                                    0.5, 2.5,
                                    1.5, 2.5,
```

```
                                    2.5, 2.5,
                                    3.5, 2.5,
                                    0.5, 1.5,
                                    1.5, 1.5,
                                    2.5, 1.5,
                                    3.5, 1.5,
                                    0.5, 0.5,
                                    1.5, 0.5,
                                    2.5, 0.5,
                                    3.5, 0.5), ncol=2,byrow = TRUE)
colnames(coordsHabitatGridCenter) <- c("x","y")
# CREATE OBSERVATION WINDOWS
trapCoords <- matrix(c(1.5, 1.5, 2.5, 1.5, 1.5, 2.5, 2.5, 2.5), nrow = 4, byrow = TRUE)
colnames(trapCoords) <- c("x","y")
# PLOT CHECK
plot(coordsHabitatGridCenter[,"y"]~coordsHabitatGridCenter[,"x"],pch=16)
points(trapCoords[,"y"]~trapCoords[,"x"],col="red",pch=16)

# PARAMETERS
p0 <- 0.2
sigma <- 2
indicator <- 1
# WE CONSIDER 2 INDIVIDUALS
y <- matrix(c(0, 1, 1, 0,
              0, 1, 0, 1),ncol=4,nrow=2)
s <- matrix(c(0.5, 1,
              1.6, 2.3),ncol=2,nrow=2)

# RESCALE COORDINATES
ScaledtrapCoords <- scaleCoordsToHabitatGrid(coordsData =  trapCoords,
                                     coordsHabitatGridCenter = coordsHabitatGridCenter)
ScaledtrapCoords<- ScaledtrapCoords$coordsDataScaled
habitatMask <- matrix(1, nrow = 4, ncol=4, byrow = TRUE)


# CREATE LOCAL OBJECTS
TrapLocal <- getLocalObjects(habitatMask = habitatMask,
                                    coords = ScaledtrapCoords,
                                    dmax=2.5,
                                    resizeFactor = 1,
                                    plot.check = TRUE
)

# GET SPARSE MATRIX
SparseY <- getSparseY(y)

# II. USING THE DENSITY FUNCTION
 # WE TAKE THE FIRST INDIVIDUAL
i=1
  # OPTION 1: USING THE RANDOM GENERATION FUNCTIONNALITY
dbinomLocal_normal(x=SparseY$y[i,,1],
                   detNums=SparseY$detNums[i],
                   detIndices=SparseY$detIndices[i,,1],
```

```
                    size=rep(1,4),
                    p0 = p0,
                    sigma= sigma,
                    s=s[i,1:2],
                    trapCoords=ScaledtrapCoords,
                    localTrapsIndices=TrapLocal$localIndices,
                    localTrapsNum=TrapLocal$numLocalIndices,
                    resizeFactor=TrapLocal$resizeFactor,
                    habitatGrid=TrapLocal$habitatGrid,
                    indicator=indicator)

  # OPTION 2: USING RANDOM GENERATION FUNCTIONNALITY
  # WE DO NOT PROVIDE THE detNums AND detIndices ARGUMENTS
dbinomLocal_normal(x=SparseY$yCombined[i,,1],
                    size=rep(1,4),
                    p0 = p0,
                    sigma= sigma,
                    s=s[i,1:2],
                    trapCoords=ScaledtrapCoords,
                    localTrapsIndices=TrapLocal$localIndices,
                    localTrapsNum=TrapLocal$numLocalIndices,
                    resizeFactor=TrapLocal$resizeFactor,
                    habitatGrid=TrapLocal$habitatGrid,
                    indicator=indicator,
                    lengthYCombined = SparseY$lengthYCombined)

# III. USING THE RANDOM GENERATION FUNCTION
rbinomLocal_normal(n=1,
                    size=rep(1,4),
                    p0 = p0,
                    sigma= sigma,
                    s=s[i,1:2],
                    trapCoords=ScaledtrapCoords,
                    localTrapsIndices=TrapLocal$localIndices,
                    localTrapsNum=TrapLocal$numLocalIndices,
                    resizeFactor=TrapLocal$resizeFactor,
                    habitatGrid=TrapLocal$habitatGrid,
                    indicator=indicator,
                    lengthYCombined = SparseY$lengthYCombined)
```

`dbinomLocal_normalPlateau`

Local evaluation of a binomial SCR observation process

Description

The `dbinomLocal_normalPlateau` distribution is a NIMBLE custom distribution which can be used to model and simulate binomial observations (x) of a single individual over a set of traps defined by their coordinates *trapCoords* the distribution assumes that an individual's detection probability at any trap follows a half-normal plateau function of the distance between the individual's

activity center (s) and the trap location. With the half-normal plateau function, detection probability remains constant with value p0 for a plateau of width w before declining with scale sigma.

Usage

```
dbinomLocal_normalPlateau(
  x,
  detNums = -999,
  detIndices,
  size,
  p0 = -999,
  p0Traps,
  sigma,
  w = 2,
  s,
  trapCoords,
  localTrapsIndices,
  localTrapsNum,
  resizeFactor = 1,
  habitatGrid,
  indicator,
  lengthYCombined = 0,
  log = 0
)

rbinomLocal_normalPlateau(
  n = 1,
  detNums = -999,
  detIndices,
  size,
  p0 = -999,
  p0Traps,
  sigma,
  w = 2,
  s,
  trapCoords,
  localTrapsIndices,
  localTrapsNum,
  resizeFactor = 1,
  habitatGrid,
  indicator,
  lengthYCombined = 0
)
```

Arguments

x	Vector of individual detection frequencies. This argument can be provided in two formats: (i) with the *y* object as returned by `getSparseY`; (ii) with the *yCombined* object as returned by `getSparseY` Note that when the random gener-

ation functionality is used (rbinomLocal_normal), only the yCombined format can be used. The *yCombined* object combines *detNums*, *x*, and *detIndices* (in that order). When such consolidated representation of the detection data x is used, *detIndices* and *detNums* arguments should not be specified.

`detNums` Number of traps with at least one detection recorded in *x*; from the *detNums* object returned by the `getSparseY` function. This argument should not be specified when the *yCombined* object (returned by `getSparseY`) is provided as *x* and when detection data are simulated.

`detIndices` Vector of indices of traps where the detections in *x* were recorded; from the *detIndices* object returned by the `getSparseY` function. This argument should not be specified when x is provided as the *yCombined* object (returned by `getSparseY`) and when detection data are simulated.

`size` Vector of the number of trials (zero or more) for each trap (*trapCoords*).

`p0` Baseline detection probability (scalar) used in the half-normal detection function. For trap-specific baseline detection probabilities use argument *p0Traps* (vector) instead.

`p0Traps` Vector of baseline detection probabilities for each trap used in the half-normal detection function. When *p0Traps* is used, *p0* should not be provided.

`sigma` Scale parameter of the half-normal detection function.

`w` Length of plateau of the half-normal plateau detection function.

`s` Individual activity center x- and y-coordinates scaled to the habitat (see (`scaleCoordsToHabitatGrid`).

`trapCoords` Matrix of x- and y-coordinates of all traps scaled to the habitat (see (`scaleCoordsToHabitatGrid`).

`localTrapsIndices` Matrix of indices of local traps around each habitat grid cell, as returned by the `getLocalObjects` function.

`localTrapsNum` Vector of numbers of local traps around all habitat grid cells, as returned by the `getLocalObjects` function.

`resizeFactor` Aggregation factor used in the `getLocalObjects` function to reduce the number of habitat grid cells to retrieve local traps for.

`habitatGrid` Matrix of local habitat grid cell indices, from *habitatGrid* returned by the `getLocalObjects` function.

`indicator` Binary argument specifying whether the individual is available for detection (indicator = 1) or not (indicator = 0).

`lengthYCombined` The length of the x argument when the (*yCombined*) format of the detection data is provided; from the *lengthYCombined* object returned by `getSparseY`

`log` Logical argument, specifying whether to return the log-probability of the distribution.

`n` Integer specifying the number of realizations to generate. Only n = 1 is supported.

Details

All coordinates (s and trapCoords) should be scaled to the habitat (see scaleCoordsToHabitatGrid)

The dbinomLocal_normalPlateau distribution incorporates three features to increase computation efficiency (see Turek et al., 2021 <doi.org/10.1002/ecs2.3385> for more details):

1. A local evaluation of the detection probability calculation (see Milleret et al., 2019 <doi:10.1002/ece3.4751> for more details)
2. A sparse matrix representation (*x*, *detIndices* and *detNums*) of the observation data to reduce the size of objects to be processed.
3. An indicator (*indicator*) to shortcut calculations for individuals unavailable for detection.

The dbinomLocal_normalPlateau distribution requires x- and y- detector coordinates (*trapCoords*) to be scaled to the habitat grid (*habitatGrid*) using the (scaleCoordsToHabitatGrid function.)

When the aim is to simulate detection data:

1. *x* should be provided using the *yCombined* object as returned by getSparseY,
2. arguments *detIndices* and *detNums* should not be provided,
3. argument *lengthYCombined* should be provided using the *lengthYCombined* object as returned by getSparseY.

Value

The log-likelihood value associated with the vector of detections, given the location of the activity center (s), and the half-normal plateau detection function : $p = p0$ when $d < w$ and $p = p0 * exp(-(d-w)^2/\sigma^2)$ when $d >= w$.

Author(s)

Soumen Dey

References

Dey, S., Bischof, R., Dupont, P. P. A., & Milleret, C. (2022). Does the punishment fit the crime? Consequences and diagnosis of misspecified detection functions in Bayesian spatial capture–recapture modeling. Ecology and Evolution, 12, e8600. https://doi.org/10.1002/ece3.8600

Examples

```
# A user friendly vignette is also available on github:
# https://github.com/nimble-dev/nimbleSCR/blob/master/nimbleSCR/vignettes/
# Vignette name: Fit_with_dbinomLocal_normalPlateau_and_HomeRangeAreaComputation.rmd

# I. DATA SET UP
coordsHabitatGridCenter <- matrix(c(0.5, 3.5,
                                    1.5, 3.5,
                                    2.5, 3.5,
                                    3.5, 3.5,
                                    0.5, 2.5,
```

```
                                    1.5, 2.5,
                                    2.5, 2.5,
                                    3.5, 2.5,
                                    0.5, 1.5,
                                    1.5, 1.5,
                                    2.5, 1.5,
                                    3.5, 1.5,
                                    0.5, 0.5,
                                    1.5, 0.5,
                                    2.5, 0.5,
                                    3.5, 0.5), ncol=2,byrow = TRUE)
colnames(coordsHabitatGridCenter) <- c("x","y")
# CREATE OBSERVATION WINDOWS
trapCoords <- matrix(c(1.5, 1.5, 2.5, 1.5, 1.5, 2.5, 2.5, 2.5), nrow = 4, byrow = TRUE)
colnames(trapCoords) <- c("x","y")
# PLOT CHECK
plot(coordsHabitatGridCenter[,"y"]~coordsHabitatGridCenter[,"x"],pch=16)
points(trapCoords[,"y"]~trapCoords[,"x"],col="red",pch=16)

# PARAMETERS
p0 <- 0.25
sigma <- 1
w <- 1.5
indicator <- 1
# WE CONSIDER 2 INDIVIDUALS
y <- matrix(c(0, 1, 1, 0,
              0, 1, 0, 1),ncol=4,nrow=2)
s <- matrix(c(0.5, 1,
              1.6, 2.3),ncol=2,nrow=2)

# RESCALE COORDINATES
ScaledtrapCoords <- scaleCoordsToHabitatGrid(coordsData =  trapCoords,
                                     coordsHabitatGridCenter = coordsHabitatGridCenter)
ScaledtrapCoords<- ScaledtrapCoords$coordsDataScaled
habitatMask <- matrix(1, nrow = 4, ncol=4, byrow = TRUE)


# CREATE LOCAL OBJECTS
TrapLocal <- getLocalObjects(habitatMask = habitatMask,
                                   coords = ScaledtrapCoords,
                                   dmax=2.5,
                                   resizeFactor = 1,
                                   plot.check = TRUE
)

# GET SPARSE MATRIX
SparseY <- getSparseY(y)

# II. USING THE DENSITY FUNCTION
 # WE TAKE THE FIRST INDIVIDUAL
i=1
  # OPTION 1: USING THE RANDOM GENERATION FUNCTIONNALITY
dbinomLocal_normalPlateau(x=SparseY$y[i,,1],
```

```
                    detNums=SparseY$detNums[i],
                    detIndices=SparseY$detIndices[i,,1],
                    size=rep(1,4),
                    p0 = p0,
                    sigma= sigma,
                    w = w,
                    s=s[i,1:2],
                    trapCoords=ScaledtrapCoords,
                    localTrapsIndices=TrapLocal$localIndices,
                    localTrapsNum=TrapLocal$numLocalIndices,
                    resizeFactor=TrapLocal$resizeFactor,
                    habitatGrid=TrapLocal$habitatGrid,
                    indicator=indicator)

  # OPTION 2: USING RANDOM GENERATION FUNCTIONNALITY
  # WE DO NOT PROVIDE THE detNums AND detIndices ARGUMENTS
dbinomLocal_normalPlateau(x=SparseY$yCombined[i,,1],
                    size=rep(1,4),
                    p0 = p0,
                    sigma= sigma,
                    w = w,
                    s=s[i,1:2],
                    trapCoords=ScaledtrapCoords,
                    localTrapsIndices=TrapLocal$localIndices,
                    localTrapsNum=TrapLocal$numLocalIndices,
                    resizeFactor=TrapLocal$resizeFactor,
                    habitatGrid=TrapLocal$habitatGrid,
                    indicator=indicator,
                    lengthYCombined = SparseY$lengthYCombined)

# III. USING THE RANDOM GENERATION FUNCTION
rbinomLocal_normalPlateau(n=1,
                    size=rep(1,4),
                    p0 = p0,
                    sigma= sigma,
                    w = w,
                    s=s[i,1:2],
                    trapCoords=ScaledtrapCoords,
                    localTrapsIndices=TrapLocal$localIndices,
                    localTrapsNum=TrapLocal$numLocalIndices,
                    resizeFactor=TrapLocal$resizeFactor,
                    habitatGrid=TrapLocal$habitatGrid,
                    indicator=indicator,
                    lengthYCombined = SparseY$lengthYCombined)
```

---

dbinom_vector *Vectorized binomial distribution*

Description

The dbinom_vector distribution is a vectorized version of the binomial distribution. It can be used to model a vector of binomial realizations. NB: using the vectorized version is beneficial only when the entire joint likelihood of the vector of binomial realizations (x) is calculated simultaneously.

Usage

```
dbinom_vector(x, size, prob, log = 0)

rbinom_vector(n = 1, size, prob)
```

Arguments

x	Vector of quantiles.
size	Vector of number of trials (zero or more).
prob	Vector of success probabilities on each trial
log	Logical argument, specifying whether to return the log-probability of the distribution.
n	Number of observations. Only n = 1 is supported.

Value

The log-likelihood value associated with the vector of binomial observations.

Author(s)

Pierre Dupont

Examples

```
## define vectorized model code
code <- nimbleCode({
    p ~ dunif(0,1)
    p_vector[1:J] <- p
    y[1:J] ~ dbinom_vector(size = trials[1:J],
                          prob = p_vector[1:J])
})

## simulate binomial data
J <- 1000
trials <- sample(x = 10, size = J, replace = TRUE)
y <- rbinom_vector(J, size = trials, prob = 0.21)

constants <- list(J = J, trials = trials)

data <- list(y = y)

inits <- list(p = 0.5)
```

```
## create NIMBLE model object
Rmodel <- nimbleModel(code, constants, data, inits)

## use model object for MCMC, etc.
```

`dcatState1Alive1Dead` *Density and random generation of a categorical distribution describing state transition with one alive and one dead states.*

Description

The `dcatState1Alive1Dead` distribution is a NIMBLE custom distribution which can be used to model and simulate individual state transition. This function can be used to model transitions from one alive and one dead state. If z_i,t = 1, individual i can be recruited (transition to state 2) with probability prob1To2_t, so z_i,t+1 ~ dcat(1- prob1To2_t, prob1To2_t, 0,0 , 0) where prob1To2_t represent the probability of an unborn individual to be recruited. If z_i,t = 2, individual i can die and transition to z_i,t+1=3 with probability prob2To3, or survive with probability 1-prob2To3 Individuals in dead states (z_i,t = 3) remain in that state with probability 1, the absorbing state. If transition probabilities are spatially variable, a probability vector containing the transition probability value in each habitat window can be provided using the "Hab" arguments (e.g. prob1To2Hab,prob2To3Hab).

Usage

```
dcatState1Alive1Dead(
  x,
  z,
  prob1To2 = -999,
  prob1To2Hab,
  prob2To3 = -999,
  prob2To3Hab,
  s,
  habitatGrid,
  log = 0
)

rcatState1Alive1Dead(
  n,
  z,
  prob1To2 = -999,
  prob1To2Hab,
  prob2To3 = -999,
  prob2To3Hab,
  s,
  habitatGrid
)
```

Arguments

x	Scalar, individual state z_i,t+1.
z	Scalar, initial individual state z_i,t.
prob1To2	scalar, probability to transition from state 1 to 2.
prob1To2Hab	vector, Spatially-explicit probability to transition from state 2 to 3. The length of the vector should be equal the number of habitat windows in habitatGrid.
prob2To3	scalar, probability to transition from state 2 to 3.
prob2To3Hab	vector, Spatially-explicit probability to transition from state 2 to 3. The length of the vector should be equal the number of habitat windows in habitatGrid.
s	Vector of x- and y-coordinates corresponding to the AC location of the individual. Used to extract transition spatially-explicit probabilities when they are provided.
habitatGrid	Matrix of habitat window indices. Cell values should correspond to the order of habitat windows in prob1To2Hab, prob2To3Hab and in lowerCoords and upperCoords as used in the dbernppAC function.
log	Logical argument, specifying whether to return the log-probability of the distribution. dcatState1Alive1Dead gives the (log) probability density of x. rcatState1Alive2Dead gives a randomly generated individual states conditional on the initial state z.
n	Integer specifying the number of realizations to generate. Only n = 1 is supported.

Author(s)

Cyril Milleret

Examples

```
# Use the distribution in R
z <- 2
prob1To2 <- 0.2
prob2To3 <- 0.7

lowerCoords <- matrix(c(0, 0, 1, 0, 0, 1, 1, 1), nrow = 4, byrow = TRUE)
upperCoords <- matrix(c(1, 1, 2, 1, 1, 2, 2, 2), nrow = 4, byrow = TRUE)
logIntensities <- log(rep(1,4))
logSumIntensity <- log(sum(c(1:4)))
habitatGrid <- matrix(c(1:4), nrow = 2, byrow = TRUE)
numGridRows <- nrow(habitatGrid)
numGridCols <- ncol(habitatGrid)
s <- rbernppAC(n=1, lowerCoords, upperCoords, logIntensities, logSumIntensity,
               habitatGrid, numGridRows, numGridCols)

## No spatial mortality
zPlusOne <- rcatState1Alive1Dead( z = z
                                , prob1To2 = prob1To2
                                , prob2To3 = prob2To3
                                , s = s
```

```
                                  , habitatGrid = habitatGrid)
zPlusOne

dcatState1Alive1Dead(  x = zPlusOne
                     , z = z
                     , prob1To2 = prob1To2
                     , prob2To3 = prob2To3
                     , s = s
                     , habitatGrid = habitatGrid)

##  With spatial mortality
prob2To3Hab <- c(0.60, 0.70, 0.74, 0.65)
prob1To2Hab <- c(0.4,0.5,0.1,0.3)
zPlusOne <- rcatState1Alive1Dead( z = z
                                , prob1To2Hab = prob1To2Hab
                                , prob2To3Hab = prob2To3Hab
                                , s = s
                                , habitatGrid = habitatGrid)
zPlusOne
dcatState1Alive1Dead(  x = zPlusOne
                     , z = z
                     , prob1To2Hab = prob1To2Hab
                     , prob2To3Hab = prob2To3Hab
                     , s = s
                     , habitatGrid = habitatGrid)
```

`dcatState1Alive2Dead` *Density and random generation of a categorical distribution describing state transition with one alive and two dead states.*

Description

The `dcatState1Alive2Dead` distribution is a NIMBLE custom distribution which can be used to model and simulate individual state transition. This function can be used to model transitions from one alive and two dead states. If z_i,t = 1, individual i can be recruited (transition to state 2) with probability prob1To2_t, so z_i,t+1 ~ dcat(1- prob1To2_t, prob1To2_t, 0,0 , 0) where prob1To2_t represent the probability of an unborn individual to be recruited. If z_i,t = 2, individual i can die from one cause of mortality (e.g. culling) and transition to z_i,t+1=3 with probability prob2To3, or die from another cause with probability prob2To4 z_i,t+1=4. If the individual does not die it can survive and remain in state 2 with probability (1-(prob2To3+prob2To4)). Individuals in dead states (z_i,t = 3 or 4) transition to z_i,t+1 = 4, the absorbing state, with probability 1. If transition probabilities are spatially variable, a probability vector containing the transition probability value in each habitat window can be provided using the "Hab" arguments (e.g. prob1To2Hab,prob2To3Hab).

Usage

```
dcatState1Alive2Dead(
  x,
  z,
  prob1To2 = -999,
  prob1To2Hab,
  prob2To3 = -999,
  prob2To3Hab,
  prob2To4 = -999,
  prob2To4Hab,
  s,
  habitatGrid,
  log = 0
)

rcatState1Alive2Dead(
  n,
  z,
  prob1To2 = -999,
  prob1To2Hab,
  prob2To3 = -999,
  prob2To3Hab,
  prob2To4 = -999,
  prob2To4Hab,
  s,
  habitatGrid
)
```

Arguments

x	Scalar, individual state z_i,t+1.
z	Scalar, initial individual state z_i,t.
prob1To2	scalar, probability to transition from state 1 to 2.
prob1To2Hab	vector, Spatially-explicit probability to transition from state 2 to 3. The length of the vector should be equal the number of habitat windows in habitatGrid.
prob2To3	scalar, probability to transition from state 2 to 3.
prob2To3Hab	vector, Spatially-explicit probability to transition from state 2 to 3. The length of the vector should be equal the number of habitat windows in habitatGrid.
prob2To4	scalar, probability to transition from state 2 to 4.
prob2To4Hab	vector, Spatially-explicit probability to transition from state 2 to 4. The length of the vector should be equal the number of habitat windows in habitatGrid.
s	Vector of x- and y-coordinates corresponding to the AC location of the individual. Used to extract transition spatially-explicit probabilities when they are provided.

habitatGrid	Matrix of habitat window indices. Cell values should correspond to the order of habitat windows in prob1To2Hab, prob2To3Hab and in lowerCoords and upperCoords as used in the dbernppAC function.
log	Logical argument, specifying whether to return the log-probability of the distribution.
n	Integer specifying the number of realizations to generate. Only n = 1 is supported.

Value

dcatState1Alive2Dead gives the (log) probability density of x. rcatState1Alive2Dead gives a randomly generated individual states conditional on the initial state z.

Author(s)

Cyril Milleret

Examples

```
# Use the distribution in R

z <- 2
prob1To2 <- 0.2
prob2To3 <- 0.4
prob2To4 <- 0.1

lowerCoords <- matrix(c(0, 0, 1, 0, 0, 1, 1, 1), nrow = 4, byrow = TRUE)
upperCoords <- matrix(c(1, 1, 2, 1, 1, 2, 2, 2), nrow = 4, byrow = TRUE)
logIntensities <- log(rep(1,4))
logSumIntensity <- log(sum(c(1:4)))
habitatGrid <- matrix(c(1:4), nrow = 2, byrow = TRUE)
numGridRows <- nrow(habitatGrid)
numGridCols <- ncol(habitatGrid)
s <- rbernppAC(n=1, lowerCoords, upperCoords, logIntensities, logSumIntensity,
                habitatGrid, numGridRows, numGridCols)

## No spatial mortality
zPlusOne <- rcatState1Alive2Dead( z = z
                                , prob1To2 = prob1To2
                                , prob2To3 = prob2To3
                                , prob2To4 = prob2To4
                                , s = s
                                , habitatGrid = habitatGrid)

dcatState1Alive2Dead(  x = zPlusOne
                     , z = z
                     , prob1To2 = prob1To2
                     , prob2To3 = prob2To3
                     , prob2To4 = prob2To4
                     , s = s
                     , habitatGrid = habitatGrid)
```

```
##  With spatial mortality
prob2To3Hab <- c(0.10, 0.20, 0.15, 0.30)
prob2To4Hab <- c(0.13, 0.21, 0.12, 0.08)
phiSpatial <- 1-(prob2To3Hab+prob2To4Hab)
zPlusOne <- rcatState1Alive2Dead( z = z
                                , prob1To2Hab = prob1To2Hab
                                , prob2To3Hab =  prob2To3Hab
                                , prob2To4Hab = prob2To4Hab
                                , s = s
                                , habitatGrid = habitatGrid)

dcatState1Alive2Dead(  x = zPlusOne
                     , z = z
                     , prob1To2Hab = prob1To2Hab
                     , prob2To3Hab =  prob2To3Hab
                     , prob2To4Hab = prob2To4Hab
                     , s = s
                     , habitatGrid = habitatGrid)
```

`dcatState2Alive2Dead` *Density and random generation of a categorical distribution describing state transition with two alive and two dead states.*

Description

The `dcatState2Alive2Dead` distribution is a NIMBLE custom distribution which can be used to model and simulate individual state transition. This function can be used to model transitions from two alive and two dead states. If z_i,t = 1, individual i can be recruited (transition to state 2) with probability prob1To2_t, so z_i,t+1 ~ dcat(1- prob1To2_t, prob1To2_t, 0,0 , 0) where prob1To2_t represent the probability of an unborn individual to be recruited. If z_i,t = 2, individual i can die from one cause of mortality (e.g. culling) and transition to z_i,t+1=4 with probability prob2To4, or die from another cause with probability prob2To5 z_i,t+1=5. If the individual does not die (1-(prob2To4+prob2To5)), it can either transition to the second state alive (z_i,t+1=3) with probability prob2To3 or remain in the first state alive (z_i,t+1=2) with probability (1-prob2To3). If z_i,t = 3, individual i can die from one cause of mortality (e.g. culling) and transition to z_i,t+1=4 with probability prob3To4, or die from another cause with probability prob3To5 z_i,t+1=5. if the individual does not die (1-(prob3To4+prob3To5)), the individual remain in state 3. Individuals in dead states (z_i,t = 4 or 5) transition to z_i,t+1 = 5, the absorbing state, with probability 1. If transition probabilities are spatially variable, a probability vector containing the transition probability value in each habitat window can be provided using the "Hab" arguments (e.g. prob1To2Hab,prob2To3Hab).

Usage

```
dcatState2Alive2Dead(
```

```
  x,
  z,
  prob1To2 = -999,
  prob1To2Hab,
  prob2To3 = -999,
  prob2To3Hab,
  prob2To4 = -999,
  prob2To4Hab,
  prob3To4 = -999,
  prob3To4Hab,
  prob2To5 = -999,
  prob2To5Hab,
  prob3To5 = -999,
  prob3To5Hab,
  s,
  habitatGrid,
  log = 0
)

rcatState2Alive2Dead(
  n,
  z,
  prob1To2 = -999,
  prob1To2Hab,
  prob2To3 = -999,
  prob2To3Hab,
  prob2To4 = -999,
  prob2To4Hab,
  prob3To4 = -999,
  prob3To4Hab,
  prob2To5 = -999,
  prob2To5Hab,
  prob3To5 = -999,
  prob3To5Hab,
  s,
  habitatGrid
)
```

Arguments

x	Scalar, individual state z_i,t+1.
z	Scalar, initial individual state z_i,t.
prob1To2	scalar, probability to transition from state 1 to 2.
prob1To2Hab	vector, Spatially-explicit probability to transition from state 2 to 3. The length of the vector should be equal the number of habitat windows in habitatGrid.
prob2To3	scalar, probability to transition from state 2 to 3.
prob2To3Hab	vector, Spatially-explicit probability to transition from state 2 to 3. The length of the vector should be equal the number of habitat windows in habitatGrid.

prob2To4	scalar, probability to transition from state 2 to 4.
prob2To4Hab	vector, Spatially-explicit probability to transition from state 2 to 4. The length of the vector should be equal the number of habitat windows in habitatGrid.
prob3To4	scalar, probability to transition from state 3 to 4.
prob3To4Hab	vector, Spatially-explicit probability to transition from state 3 to 4. The length of the vector should be equal the number of habitat windows in habitatGrid.
prob2To5	scalar, probability to transition from state 2 to 5.
prob2To5Hab	vector, Spatially-explicit probability to transition from state 2 to 5. The length of the vector should be equal the number of habitat windows in habitatGrid.
prob3To5	scalar, probability to transition from state 3 to 5.
prob3To5Hab	vector, Spatially-explicit probability to transition from state 3 to 5. The length of the vector should be equal the number of habitat windows in habitatGrid.
s	Vector of x- and y-coordinates corresponding to the AC location of the individual. Used to extract transition spatially-explicit probabilities when they are provided.
habitatGrid	Matrix of habitat window indices. Cell values should correspond to the order of habitat windows in prob1To2Hab, prob2To3Hab and in lowerCoords and upperCoords as used in the dbernppAC function.
log	Logical argument, specifying whether to return the log-probability of the distribution.
n	Integer specifying the number of realizations to generate. Only n = 1 is supported.

Value

dcatState2Alive2Dead gives the (log) probability density of x. dcatState2Alive2Dead gives a randomly generated individual states conditional on the initial state z.

Author(s)

Cyril Milleret

Examples

```
# Use the distribution in R

z <- 3
prob1To2 <- 0.2
prob2To3 <- 0.4
prob2To4 <- 0.1
prob2To5 <- 0.1

prob3To4 <- 0.2
prob3To5 <- 0.1


lowerCoords <- matrix(c(0, 0, 1, 0, 0, 1, 1, 1), nrow = 4, byrow = TRUE)
```

```
upperCoords <- matrix(c(1, 1, 2, 1, 1, 2, 2, 2), nrow = 4, byrow = TRUE)
logIntensities <- log(rep(1,4))
logSumIntensity <- log(sum(c(1:4)))
habitatGrid <- matrix(c(1:4), nrow = 2, byrow = TRUE)
numGridRows <- nrow(habitatGrid)
numGridCols <- ncol(habitatGrid)
s <- rbernppAC(n=1, lowerCoords, upperCoords, logIntensities, logSumIntensity,
              habitatGrid, numGridRows, numGridCols)

## No spatial mortality
zPlusOne <- rcatState2Alive2Dead( z = z
                                , prob1To2 = prob1To2
                                , prob2To3 = prob2To3
                                , prob2To4 = prob2To4
                                , prob2To5 = prob2To5
                                , prob3To4 = prob3To4
                                , prob3To5 = prob3To5
                                , s = s
                                , habitatGrid = habitatGrid)

dcatState2Alive2Dead(  x = zPlusOne
                     , z = z
                     , prob1To2 = prob1To2
                     , prob2To3 = prob2To3
                     , prob2To4 = prob2To4
                     , prob2To5 = prob2To5
                     , prob3To4 = prob3To4
                     , prob3To5 = prob3To5
                     , s = s
                     , habitatGrid = habitatGrid)

##  With spatial mortality
prob2To3Hab <- runif(length(habitatGrid),0,0.1)
prob2To4Hab <- runif(length(habitatGrid),0,0.1)
prob2To5Hab <- runif(length(habitatGrid),0,0.1)
prob3To4Hab <- runif(length(habitatGrid),0,0.1)
prob3To5Hab <- runif(length(habitatGrid),0,0.1)



zPlusOne <- rcatState2Alive2Dead( z = z
                                , prob1To2 = prob1To2
                                , prob2To3Hab = prob2To3Hab
                                , prob2To4Hab = prob2To4Hab
                                , prob2To5Hab = prob2To5Hab
                                , prob3To4Hab = prob3To4Hab
                                , prob3To5Hab = prob3To5Hab
                                , s = s
                                , habitatGrid = habitatGrid)

dcatState2Alive2Dead(  x = zPlusOne
                     , z = z
                     , prob1To2 = prob1To2
```

```
, prob2To3Hab = prob2To3Hab
, prob2To4Hab = prob2To4Hab
, prob2To5Hab = prob2To5Hab
, prob3To4Hab = prob3To4Hab
, prob3To5Hab = prob3To5Hab
, s = s
, habitatGrid = habitatGrid)
```

`dDispersal_exp` *Bivariate exponential dispersal distribution for activity centers*

Description

This function is deprecated, and it will be removed from a future release.

Usage

```
dDispersal_exp(x, s, rate, log)

rDispersal_exp(n, s, rate)
```

Arguments

`x`	Bivariate activity center coordinates (at time t+1).
`s`	Current location of the bivariate activity center (at time t).
`rate`	Rate parameter of the exponential distribution for dispersal distance.
`log`	Logical argument, specifying whether to return the log-probability of the distribution.
`n`	Integer specifying the number of realisations to generate. Only n = 1 is supported.

Details

The dDispersal_exp distribution is a bivariate distribution which can be used to model the latent bivariate activity centers (ACs) of individuals in a population. This distribution models the situation when individual AC dispersal is uniform in direction (that is, dispersal occurs in a direction theta, where theta is uniformly distributed on [-pi, pi]), and with an exponential distribution for the radial dispersal distance.

The dDispersal_exp distribution models the location of an AC at time (t+1), conditional on the previous AC location at time (t) and the rate parameter (rate) of the exponential distribution for dispersal distance.

Value

The log-probability value associated with the bivariate activity center location x, given the current activity center s, and the rate parameter of the exponential dispersal distance distribution.

Author(s)

Daniel Turek

Examples

```
## Not run:

## define model code
code <- nimbleCode({
    lambda ~ dgamma(0.001, 0.001)
    for(i in 1:N) {
        AC[i, 1, 1] ~ dunif(0, 100)
        AC[i, 2, 1] ~ dunif(0, 100)
        for(t in 2:T) {
            AC[i, 1:2, t+1] ~ dDispersal_exp(s = AC[i, 1:2, t], rate = lambda)
        }
    }
})

constants <- list(N = 10, T = 6)

## create NIMBLE model object
Rmodel <- nimbleModel(code, constants)

## use model object for MCMC, etc.


## End(Not run)
```

---

dHabitatMask *Ones trick distribution for irregular habitat shapes*

Description

The dHabitatMask distribution checks and ensures that the proposed activity center location (s) falls within the suitable habitat (defined in the binary matrix habitatMask).

Usage

```
dHabitatMask(x, s, xmax, xmin, ymax, ymin, habitatMask, log = 0)

rHabitatMask(n, s, xmax, xmin, ymax, ymin, habitatMask)
```

Arguments

x	Ones trick data.
s	Bivariate activity center coordinates.

xmax	Maximum of trap location x-coordinates.
xmin	Minimum of trap location x-coordinates.
ymax	Maximum of trap location y-coordinates.
ymin	Minimum of trap location y-coordinates.
habitatMask	A binary matrix object indicating which cells are considered as suitable habitat.
log	Logical argument, specifying whether to return the log-probability of the distribution.
n	Integer specifying the number of realisations to generate. Only n = 1 is supported.

Details

The rHabitatMask function returns the value of the habitat mask cell (0 or 1) where the proposed activity center falls. See also M. Meredith: SECR in BUGS/JAGS with patchy habitat.

Value

The log-likelihood value associated with the bivariate activity center location s being in the suitable habitat (i.e. 0 if it falls within the habitat mask and -Inf otherwise).

Author(s)

Daniel Turek

Examples

```
## define model code
code <- nimbleCode({
    for(i in 1:N) {
        s[i, 1] ~ dunif(0, 100)
        s[i, 2] ~ dunif(0, 100)
        OK[i] ~ dHabitatMask( s = s[i,1:2],
                              xmax = 100,
                              xmin = 0,
                              ymax = 100,
                              ymin = 0,
                              habitatMask = habitatMask[1:100,1:100])
    }
})

N <- 20

habitatMask <- matrix(rbinom(10000,1,0.75), nrow = 100)

constants <- list(N = N, habitatMask = habitatMask)

data <- list(OK = rep(1, N))

inits <- list(s = array(runif(2*N, 0, 100), c(N,2)))
```

```
## create NIMBLE model object
Rmodel <- nimbleModel(code, constants, data, inits)

## use model object for MCMC, etc.
```

---

dmultiLocal_normal *Local evaluation of a multinomial SCR detection process*

Description

The dmultiLocal_normal distribution is a NIMBLE custom distribution which can be used to model and simulate multinomial observations (x) of a single individual over a set of traps defined by their coordinates *trapCoords* the distribution assumes that an individual's detection probability at any trap follows a half-normal function of the distance between the individual's activity center (s) and the trap location. All coordinates (s and trapCoords) should be scaled to the habitat (see scaleCoordsToHabitatGrid)

Usage

```
dmultiLocal_normal(
  x,
  detNums = -999,
  detIndices,
  size,
  p0 = -999,
  p0Traps,
  sigma,
  s,
  trapCoords,
  localTrapsIndices,
  localTrapsNum,
  resizeFactor = 1,
  habitatGrid,
  indicator,
  lengthYCombined = 0,
  log = 0
)

rmultiLocal_normal(
  n = 1,
  detNums = -999,
  detIndices,
  size,
  p0 = -999,
  p0Traps,
  sigma,
```

```
  s,
  trapCoords,
  localTrapsIndices,
  localTrapsNum,
  resizeFactor = 1,
  habitatGrid,
  indicator,
  lengthYCombined = 0
)
```

Arguments

x	Vector of individual detection frequencies. This argument can be provided in two formats: (i) with the *y* object as returned by the getSparseY function; (ii) with the *yCombined* object as returned by getSparseY. Note that when the random generation functionality is used (rmultiLocal_normal), only the *yCombined* format can be used. The *yCombined* object combines *detNums*, *x*, and *detIndices* (in that order). When such consolidated representation of the detection data *x* is used, *detIndices* and *detNums* arguments should not be specified.
detNums	umber of traps with at least one detection recorded in *x*; from the *detNums* object returned by the getSparseY function. This argument should not be specified when the *yCombined* object (returned by getSparseY) is provided as *x* and when detection data are simulated.
detIndices	Vector of indices of traps where the detections in *x* were recorded; from the *detIndices* object returned by the getSparseY function. This argument should not be specified when x is provided as the *yCombined* object (returned by getSparseY) and when detection data are simulated.
size	Number of occasions.
p0	Baseline detection probability (scalar) used in the half-normal detection function. For trap-specific baseline detection probabilities use argument *p0Traps* (vector) instead.
p0Traps	Vector of baseline detection probabilities for each trap used in the half-normal detection function. When *p0Traps* is used, *p0* should not be provided.
sigma	Scale parameter of the half-normal detection function.
s	Individual activity center x- and y-coordinates scaled to the habitat (see (scaleCoordsToHabitatGrid)).
trapCoords	Matrix of x- and y-coordinates of all traps scaled to the habitat (see (scaleCoordsToHabitatGrid)).
localTrapsIndices	Matrix of indices of local traps around each habitat grid cell, as returned by the getLocalObjects function.
localTrapsNum	Vector of numbers of local traps around all habitat grid cells, as returned by the getLocalObjects function.
resizeFactor	Aggregation factor used in the getLocalObjects function to reduce the number of habitat grid cells to retrieve local traps for.
habitatGrid	Matrix of local habitat grid cell indices, from *habitatGrid* returned by the getLocalObjects function.

`indicator`	Binary argument specifying whether the individual is available for detection (indicator = 1) or not (indicator = 0).
`lengthYCombined`	The length of the x argument when the (*yCombined*) format of the detection data is provided; from the *lengthYCombined* object returned by `getSparseY`
`log`	Logical argument, specifying whether to return the log-probability of the distribution.
`n`	Integer specifying the number of realizations to generate. Only n = 1 is supported.

Details

The `dmultiLocal_normal` distribution incorporates three features to increase computation efficiency (see Turek et al., 2021 <doi.org/10.1002/ecs2.3385> for more details):

1. A local evaluation of the detection probability calculation (see Milleret et al., 2019 <doi:10.1002/ece3.4751> for more details)
2. A sparse matrix representation (*x*, *detIndices* and *detNums*) of the observation data to reduce the size of objects to be processed.
3. An indicator (*indicator*) to shortcut calculations for individuals unavailable for detection.

The `dmultiLocal_normal` distribution requires x- and y- detector coordinates (*trapCoords*) and activity centers coordinates (*s*) to be scaled to the habitat grid (*habitatGrid*) using the (`scaleCoordsToHabitatGrid` function.)

When the aim is to simulate detection data:

1. *x* should be provided using the *yCombined* object as returned by `getSparseY`,
2. arguments *detIndices* and *detNums* should not be provided,
3. argument *lengthYCombined* should be provided using the *lengthYCombined* object as returned by `getSparseY`.

Value

The log-likelihood value associated with the vector of detections, given the location of the activity center (s), and the half-normal detection function : $p = p0 * exp(-d^2/2\sigma^2)$.

Author(s)

Soumen Dey, Cyril Milleret

Examples

```
# I. DATA SET UP
coordsHabitatGridCenter <- matrix(c(0.5, 3.5,
                                    1.5, 3.5,
                                    2.5, 3.5,
                                    3.5, 3.5,
                                    0.5, 2.5,
                                    1.5, 2.5,
```

```
                                    2.5, 2.5,
                                    3.5, 2.5,
                                    0.5, 1.5,
                                    1.5, 1.5,
                                    2.5, 1.5,
                                    3.5, 1.5,
                                    0.5, 0.5,
                                    1.5, 0.5,
                                    2.5, 0.5,
                                    3.5, 0.5), ncol=2,byrow = TRUE)
colnames(coordsHabitatGridCenter) <- c("x","y")
# CREATE OBSERVATION WINDOWS
trapCoords <- matrix(c(1.5, 1.5, 2.5, 1.5, 1.5, 2.5, 2.5, 2.5), nrow = 4, byrow = TRUE)
colnames(trapCoords) <- c("x","y")
# PLOT CHECK
plot(coordsHabitatGridCenter[,"y"]~coordsHabitatGridCenter[,"x"],pch=16)
points(trapCoords[,"y"]~trapCoords[,"x"],col="red",pch=16)

# PARAMETERS
p0 <- 0.2
sigma <- 2
indicator <- 1
# WE CONSIDER 2 INDIVIDUALS

y <- matrix(c(2, 1, 1,-1, 1, 4, -1,#id#1 detected 2 times at detector 1 and 4
              2, 2,-1,-1, 3,-1, -1),#id#2 detected 2 times at detector 3
               ncol=7, nrow=2, byrow = TRUE)




s <- matrix(c(0.5, 1,
              1.6, 2.3),ncol=2,nrow=2)

# RESCALE COORDINATES
ScaledtrapCoords <- scaleCoordsToHabitatGrid(coordsData =  trapCoords,
                                             coordsHabitatGridCenter = coordsHabitatGridCenter)
ScaledtrapCoords<- ScaledtrapCoords$coordsDataScaled
habitatMask <- matrix(1, nrow = 4, ncol=4, byrow = TRUE)


# CREATE LOCAL OBJECTS
TrapLocal <- getLocalObjects(habitatMask = habitatMask,
                                  coords = ScaledtrapCoords,
                                  dmax=2.5,
                                  resizeFactor = 1,
                                  plot.check = TRUE
)

# GET SPARSE MATRIX
#SparseY <- getSparseY(y)

# II. USING THE DENSITY FUNCTION
```

```
# WE TAKE THE FIRST INDIVIDUAL
i=1
 # OPTION 1: USING THE RANDOM GENERATION FUNCTIONNALITY
dmultiLocal_normal(x=y[i,]
                   ,
                   size=3
                   ,
                   p0 = p0
                   ,
                   sigma= sigma
                   ,
                   s=s[i,1:2]
                   ,
                   trapCoords=ScaledtrapCoords
                   ,
                   localTrapsIndices=TrapLocal$localIndices
                   ,
                   localTrapsNum=TrapLocal$numLocalIndices
                   ,
                   resizeFactor=TrapLocal$resizeFactor
                   ,
                   habitatGrid=TrapLocal$habitatGrid
                   ,
                   indicator=indicator
                   ,
                   lengthYCombined = ncol(y)
                   )



# III. USING THE RANDOM GENERATION FUNCTION
rmultiLocal_normal(n=1,
                   size=3,
                   p0 = p0,
                   sigma= sigma,
                   s=s[i,1:2],
                   trapCoords=ScaledtrapCoords,
                   localTrapsIndices=TrapLocal$localIndices,
                   localTrapsNum=TrapLocal$numLocalIndices,
                   resizeFactor=TrapLocal$resizeFactor,
                   habitatGrid=TrapLocal$habitatGrid,
                   indicator=indicator,
                   lengthYCombined = ncol(y))
```

---

dnormalizer *Normalizing constant generator*

Description

A normalizer used for normalizing nimble distributions. It is particularly useful for fitting dpoisppDetection_normal and dpoisppLocalDetection_normal models using the semi-complete data likelihood approach.

Usage

```
dnormalizer(x, logNormConstant, log = 0)

rnormalizer(n, logNormConstant)
```

Arguments

x	Input data, which can be any scalar and will not influence the return value.
logNormConstant	Normalizing constant on a log scale.
log	Logical. If TRUE return the log normalizing constant. Otherwise return the normalizing constant.
n	Integer specifying the number of realisations to generate. Only n = 1 is supported.

Value

The normalizing constant.

Author(s)

Wei Zhang

Examples

```
dnormalizer(1, log(0.5), log = TRUE)
dnormalizer(0, log(0.5), log = FALSE)
```

---

dpoisLocal_normal *Local evaluation of a Poisson SCR detection process*

Description

The dpoisLocal_normal distribution is a NIMBLE custom distribution which can be used to model and simulate Poisson observations (x) of a single individual over a set of traps defined by their coordinates *trapCoords* the distribution assumes that an individual's detection probability at any trap follows a half-normal function of the distance between the individual's activity center (s) and the trap location. All coordinates (s and trapCoords) should be scaled to the habitat (see scaleCoordsToHabitatGrid)

Usage

```
dpoisLocal_normal(
  x,
  detNums = -999,
  detIndices,
  lambda = -999,
  lambdaTraps,
  sigma,
  s,
  trapCoords,
  localTrapsIndices,
  localTrapsNum,
  resizeFactor = 1,
  habitatGrid,
  indicator,
  lengthYCombined = 0,
  log = 0
)

rpoisLocal_normal(
  n = 1,
  detNums = -999,
  detIndices,
  lambda = -999,
  lambdaTraps,
  sigma,
  s,
  trapCoords,
  localTrapsIndices,
  localTrapsNum,
  resizeFactor = 1,
  habitatGrid,
  indicator,
  lengthYCombined = 0
)
```

Arguments

x	Vector of individual detection frequencies. This argument can be provided in two formats: (i) with the *y* object as returned by getSparseY; (ii) with the *yCombined* object as returned by getSparseY Note that when the random generation functionality is used (rpoisLocal_normal), only the yCombined format can be used. The *yCombined* object combines *detNums*, *x*, and *detIndices* (in that order). When such consolidated representation of the detection data x is used, *detIndices* and *detNums* arguments should not be specified.
detNums	Number of traps with at least one detection recorded in *x*; from the *detNums* object returned by the getSparseY function. This argument should not be specified when the *yCombined* object (returned by getSparseY) is provided as *x* and

when detection data are simulated.

`detIndices` Vector of indices of traps where the detections in *x* were recorded; from the *detIndices* object returned by the `getSparseY` function. This argument should not be specified when x is provided as the *yCombined* object (returned by `getSparseY`) and when detection data are simulated.

`lambda` Baseline detection rate used in the half-normal detection function.

`lambdaTraps` Vector of baseline detection rate for each trap used in the half-normal detection function. When *lambdaTraps* is used, *lambda* should not be provided.

`sigma` Scale parameter of the half-normal detection function.

`s` Individual activity center x- and y-coordinates scaled to the habitat (see (`scaleCoordsToHabitatGrid`).

`trapCoords` Matrix of x- and y-coordinates of all traps scaled to the habitat (see (`scaleCoordsToHabitatGrid`).

`localTrapsIndices` Matrix of indices of local traps around each habitat grid cell, as returned by the `getLocalObjects` function.

`localTrapsNum` Vector of numbers of local traps around all habitat grid cells, as returned by the `getLocalObjects` function.

`resizeFactor` Aggregation factor used in the `getLocalObjects` function to reduce the number of habitat grid cells to retrieve local traps for.

`habitatGrid` Matrix of local habitat grid cell indices, from *habitatGrid* returned by the `getLocalObjects` function.

`indicator` Binary argument specifying whether the individual is available for detection (indicator = 1) or not (indicator = 0).

`lengthYCombined` The length of the x argument when the (*yCombined*) format of the detection data is provided; from the *lengthYCombined* object returned by `getSparseY`

`log` Logical argument, specifying whether to return the log-probability of the distribution.

`n` Integer specifying the number of realizations to generate. Only n = 1 is supported.

Details

The `dpoisLocal_normal` distribution incorporates three features to increase computation efficiency (see Turek et al., 2021 <doi.org/10.1002/ecs2.3385> for more details):

1. A local evaluation of the detection probability calculation (see Milleret et al., 2019 <doi:10.1002/ece3.4751> for more details)
2. A sparse matrix representation (*x*, *detIndices* and *detNums*) of the observation data to reduce the size of objects to be processed.
3. An indicator (*indicator*) to shortcut calculations for individuals unavailable for detection.

The `dpoisLocal_normal` distribution requires x- and y- detector coordinates (*trapCoords*) and activity centers coordinates (*s*) to be scaled to the habitat grid (*habitatGrid*) using the (`scaleCoordsToHabitatGrid` function.)

When the aim is to simulate detection data:

1. *x* should be provided using the *yCombined* object as returned by `getSparseY`,
2. arguments *detIndices* and *detNums* should not be provided,
3. argument *lengthYCombined* should be provided using the *lengthYCombined* object as returned by `getSparseY`.

Value

The log-likelihood value associated with the vector of detections, given the location of the activity center (s), and the half-normal detection function : $p = lambda * exp(-d^2/2\sigma^2)$.

Author(s)

Cyril Milleret, Soumen Dey

Examples

```
# I. DATA SET UP
coordsHabitatGridCenter <- matrix(c(0.5, 3.5,
                                    1.5, 3.5,
                                    2.5, 3.5,
                                    3.5, 3.5,
                                    0.5, 2.5,
                                    1.5, 2.5,
                                    2.5, 2.5,
                                    3.5, 2.5,
                                    0.5, 1.5,
                                    1.5, 1.5,
                                    2.5, 1.5,
                                    3.5, 1.5,
                                    0.5, 0.5,
                                    1.5, 0.5,
                                    2.5, 0.5,
                                    3.5, 0.5), ncol=2,byrow = TRUE)
colnames(coordsHabitatGridCenter) <- c("x","y")
# CREATE OBSERVATION WINDOWS
trapCoords <- matrix(c(1.5, 1.5, 2.5, 1.5, 1.5, 2.5, 2.5, 2.5), nrow = 4, byrow = TRUE)
colnames(trapCoords) <- c("x","y")
# PLOT CHECK
plot(coordsHabitatGridCenter[,"y"]~coordsHabitatGridCenter[,"x"],pch=16)
points(trapCoords[,"y"]~trapCoords[,"x"],col="red",pch=16)

# PARAMETERS
lambda <- 0.2
sigma <- 2
indicator <- 1
# WE CONSIDER 2 INDIVIDUALS
y <- matrix(c(0, 1, 1, 0,
              0, 1, 0, 1),ncol=4,nrow=2)
s <- matrix(c(0.5, 1,
              1.6, 2.3),ncol=2,nrow=2)

# RESCALE COORDINATES
```

```
ScaledtrapCoords <- scaleCoordsToHabitatGrid(coordsData =  trapCoords,
                                             coordsHabitatGridCenter = coordsHabitatGridCenter)
ScaledtrapCoords<- ScaledtrapCoords$coordsDataScaled
habitatMask <- matrix(1, nrow = 4, ncol=4, byrow = TRUE)


# CREATE LOCAL OBJECTS
TrapLocal <- getLocalObjects(habitatMask = habitatMask,
                                    coords = ScaledtrapCoords,
                                    dmax=2.5,
                                    resizeFactor = 1,
                                    plot.check = TRUE
)

# GET SPARSE MATRIX
SparseY <- getSparseY(y)

# II. USING THE DENSITY FUNCTION
 # WE TAKE THE FIRST INDIVIDUAL
i=1
  # OPTION 1: USING THE RANDOM GENERATION FUNCTIONNALITY
dpoisLocal_normal(x=SparseY$y[i,,1],
                  detNums=SparseY$detNums[i],
                  detIndices=SparseY$detIndices[i,,1],
                  lambda = lambda,
                  sigma= sigma,
                  s=s[i,1:2],
                  trapCoords=ScaledtrapCoords,
                  localTrapsIndices=TrapLocal$localIndices,
                  localTrapsNum=TrapLocal$numLocalIndices,
                  resizeFactor=TrapLocal$resizeFactor,
                  habitatGrid=TrapLocal$habitatGrid,
                  indicator=indicator)

  # OPTION 2: USING RANDOM GENERATION FUNCTIONNALITY
  # WE DO NOT PROVIDE THE detNums AND detIndices ARGUMENTS
dpoisLocal_normal(x=SparseY$yCombined[i,,1],
                  lambda = lambda,
                  sigma= sigma,
                  s=s[i,1:2],
                  trapCoords=ScaledtrapCoords,
                  localTrapsIndices=TrapLocal$localIndices,
                  localTrapsNum=TrapLocal$numLocalIndices,
                  resizeFactor=TrapLocal$resizeFactor,
                  habitatGrid=TrapLocal$habitatGrid,
                  indicator=indicator,
                  lengthYCombined = SparseY$lengthYCombined)

# III. USING THE RANDOM GENERATION FUNCTION
rpoisLocal_normal(n=1,
                  lambda = lambda,
                  sigma= sigma,
                  s=s[i,1:2],
```

```
                    trapCoords=ScaledtrapCoords,
                    localTrapsIndices=TrapLocal$localIndices,
                    localTrapsNum=TrapLocal$numLocalIndices,
                    resizeFactor=TrapLocal$resizeFactor,
                    habitatGrid=TrapLocal$habitatGrid,
                    indicator=indicator,
                    lengthYCombined = SparseY$lengthYCombined)
```

---

dpoisppAC *Poisson point process for the distribution of activity centers*

Description

Density and random generation functions of the Poisson point process for the distribution of activity centers. The dpoisppAC distribution is a NIMBLE custom distribution which can be used to model and simulate activity center locations (x) of multiple individual in continuous space over a set of habitat windows defined by their upper and lower coordinates (*lowerCoords,upperCoords*). The distribution assumes that activity centers follow a Poisson point process with intensity = exp(logIntensities). All coordinates (s and *trapCoords*) should be scaled to the habitat (scaleCoordsToHabitatGrid).

Usage

```
dpoisppAC(
  x,
  lowerCoords,
  upperCoords,
  logIntensities,
  sumIntensity,
  habitatGrid,
  numGridRows,
  numGridCols,
  numPoints,
  log = 0
)

rpoisppAC(
  n,
  lowerCoords,
  upperCoords,
  logIntensities,
  sumIntensity,
  habitatGrid,
  numGridRows,
  numGridCols,
  numPoints
)
```

Arguments

x	Matrix of x- and y-coordinates of a set of spatial points (AC locations) scaled to the habitat (scaleCoordsToHabitatGrid). Each row corresponds to a point.
lowerCoords, upperCoords	Matrices of lower and upper x- and y-coordinates of all detection windows scaled to the habitat (see (scaleCoordsToHabitatGrid). One row for each window. Each window should be of size 1x1.
logIntensities	Vector of log habitat intensities for all habitat windows.
sumIntensity	Sum of the habitat intensities over all windows. Provided as an argument for computational speed, instead of calculating it in the function.
habitatGrid	Matrix of habitat window indices. Cell values should correspond to the order of habitat windows in lowerCoords, upperCoords, and logIntensities. When the habitat grid only consists of a single row or column of windows, an additional row or column of dummy indices has to be added because the nimble model code requires a matrix.
numGridRows, numGridCols	Numbers of rows and columns of the habitat grid.
numPoints	Number of points in the Poisson point process. This value (non-negative integer) is used to truncate x so that extra rows beyond numPoints are ignored.
log	Logical argument, specifying whether to return the log-probability of the distribution.
n	Integer specifying the number of realisations to generate. Only n = 1 is supported.

Value

dpoisppAC gives the (log) probability density of the observation matrix x. rpoisppAC gives coordinates of a set of randomly generated spatial points.

Author(s)

Wei Zhang

References

W. Zhang, J. D. Chipperfield, J. B. Illian, P. Dupont, C. Milleret, P. de Valpine and R. Bischof. 2020.
A hierarchical point process model for spatial capture-recapture data. bioRxiv. DOI 10.1101/2020.10.06.325035

Examples

```
lowerCoords <- matrix(c(0, 0, 1, 0, 0, 1, 1, 1), nrow = 4, byrow = TRUE)
upperCoords <- matrix(c(1, 1, 2, 1, 1, 2, 2, 2), nrow = 4, byrow = TRUE)
logIntensities <- log(c(1:4))
logSumIntensity <- sum(exp(logIntensities))
habitatGrid <- matrix(c(1:4), nrow = 2, byrow = TRUE)
numGridRows <- nrow(habitatGrid)
numGridCols <- ncol(habitatGrid)
```

```
#Simulate data
x <- rpoisppAC(1, lowerCoords, upperCoords, logIntensities, logSumIntensity, habitatGrid,
               numGridRows, numGridCols, -1)
numPoints <- nrow(x)
dpoisppAC(x, lowerCoords, upperCoords, logIntensities, logSumIntensity,
          habitatGrid, numGridRows, numGridCols, numPoints, log = TRUE)
```

`dpoisppDetection_normal` *Poisson point process detection model*

Description

Density and random generation functions of the Poisson point process for detection. The `dpoisppDetection_normal` distribution is a NIMBLE custom distribution which can be used to model and simulate Poisson observations (x) of a single individual in continuous space over a set of detection windows defined by their upper and lower coordinates (*lowerCoords,upperCoords*). The distribution assumes that an individual's detection intensity follows an isotropic bivariate normal function centered on the individual's activity center (s) with standard deviation (*sd*). All coordinates (s and *trapCoords*) should be scaled to the habitat (`scaleCoordsToHabitatGrid`).

Usage

```
dpoisppDetection_normal(
  x,
  lowerCoords,
  upperCoords,
  s,
  sd,
  baseIntensities,
  numMaxPoints,
  numWindows,
  indicator,
  log = 0
)

rpoisppDetection_normal(
  n,
  lowerCoords,
  upperCoords,
  s,
  sd,
  baseIntensities,
  numMaxPoints,
  numWindows,
  indicator
)
```

Arguments

x	Array containing the total number of detections (x[1,1]), the x- and y-coordinates (x[2:(x[1,1]+1),1:2]), and the corresponding detection window indices (x[2:(x[1,1]+1),3]) for a set of spatial points (detection locations).
lowerCoords, upperCoords	Matrices of lower and upper x- and y-coordinates of all detection windows scaled to the habitat (see (scaleCoordsToHabitatGrid). One row for each window. Each window should be of size 1x1.
s	Vector of x- and y-coordinates of the isotropic multivariate normal distribution mean (the AC location).
sd	Standard deviation of the isotropic multivariate normal distribution.
baseIntensities	Vector of baseline detection intensities for all detection windows.
numMaxPoints	Maximum number of points. This value (non-negative integer) is only used when simulating detections to constrain the maximum number of detections.
numWindows	Number of detection windows. This value (positive integer) is used to truncate lowerCoords and upperCoords so that extra rows beyond numWindows are ignored.
indicator	Binary argument specifying whether the individual is available for detection (indicator = 1) or not (indicator = 0).
log	Logical argument, specifying whether to return the log-probability of the distribution.
n	Integer specifying the number of realisations to generate. Only n = 1 is supported.

Value

dpoisppDetection_normal gives the (log) probability density of the observation matrix x. rpoisppDetection_normal gives coordinates of a set of randomly generated spatial points.

Author(s)

Wei Zhang

References

W. Zhang, J. D. Chipperfield, J. B. Illian, P. Dupont, C. Milleret, P. de Valpine and R. Bischof. 2020.
A hierarchical point process model for spatial capture-recapture data. bioRxiv. DOI 10.1101/2020.10.06.325035

Examples

```
coordsHabitatGridCenter <- matrix(c(0.5, 3.5,
                                    1.5, 3.5,
                                    2.5, 3.5,
                                    3.5, 3.5,
                                    0.5, 2.5,
                                    1.5, 2.5,
```

```
                                    2.5, 2.5,
                                    3.5, 2.5,
                                    0.5, 1.5,
                                    1.5, 1.5,
                                    2.5, 1.5,
                                    3.5, 1.5,
                                    0.5, 0.5,
                                    1.5, 0.5,
                                    2.5, 0.5,
                                    3.5, 0.5), ncol = 2,byrow = TRUE)
colnames(coordsHabitatGridCenter) <- c("x","y")
# Create observation windows
lowerCoords <- matrix(c(0, 0, 1, 0, 0, 1, 1, 1), nrow = 4, byrow = TRUE)
upperCoords <- matrix(c(1, 1, 2, 1, 1, 2, 2, 2), nrow = 4, byrow = TRUE)
colnames(lowerCoords) <- colnames(upperCoords) <- c("x","y")

# Rescale coordinates
ScaledLowerCoords <- scaleCoordsToHabitatGrid(coordsData =  lowerCoords,
                                     coordsHabitatGridCenter = coordsHabitatGridCenter)
ScaledUpperCoords <- scaleCoordsToHabitatGrid(coordsData =  upperCoords,
                                     coordsHabitatGridCenter = coordsHabitatGridCenter)
ScaledUpperCoords$coordsDataScaled[,2] <- ScaledUpperCoords$coordsDataScaled[,2] + 1.5
ScaledLowerCoords$coordsDataScaled[,2] <- ScaledLowerCoords$coordsDataScaled[,2] - 1.5



# Detection locations
x <- matrix(c(1.5, 2, 1.1, 1.5,0.6, 2.1, 0.5, 2, 1, 1.5), nrow = 5, byrow = TRUE)

# get the window indeces on the third dimension of x

windowIndexes <- 0
for(i in 1:nrow(x)){
  windowIndexes[i] <- getWindowIndex(curCoords = x[i,],
                                     lowerCoords = ScaledLowerCoords$coordsDataScaled,
                                     upperCoords = ScaledUpperCoords$coordsDataScaled)
}
x <- cbind(x, windowIndexes)
# get the total number of detections on x[1,1]
x <- rbind(c(length(windowIndexes),0,0) ,x )


s <- c(1, 1)
sd <- 0.1
baseIntensities <- c(1:4)
windowIndices <- c(1, 2, 2, 3, 4)
numPoints <- 5
numWindows <- 4
indicator <- 1
dpoisppDetection_normal(x, lowerCoords, upperCoords, s, sd, baseIntensities,
                        numMaxPoints = dim(x)[1] , numWindows, indicator, log = TRUE)
```

`dpoisppLocalDetection_normal` *Local evaluation for a Poisson point process detection model*

Description

Density and random generation functions of the Poisson point process for detection. The `dpoisppLocalDetection_normal` distribution is a NIMBLE custom distribution which can be used to model and simulate Poisson observations (*x*) of a single individual in continuous space over a set of detection windows defined by their upper and lower coordinates (*lowerCoords,upperCoords*). The distribution assumes that an individual's detection intensity follows an isotropic bivariate normal function centered on the individual's activity center (*s*) with standard deviation (*sd*). All coordinates (*s* and *trapCoords*) should be scaled to the habitat (`scaleCoordsToHabitatGrid`).

Usage

```
dpoisppLocalDetection_normal(
  x,
  lowerCoords,
  upperCoords,
  s,
  sd,
  baseIntensities,
  habitatGridLocal,
  resizeFactor = 1,
  localObsWindowIndices,
  numLocalObsWindows,
  numMaxPoints,
  numWindows,
  indicator,
  log = 0
)

rpoisppLocalDetection_normal(
  n,
  lowerCoords,
  upperCoords,
  s,
  sd,
  baseIntensities,
  habitatGridLocal,
  resizeFactor = 1,
  localObsWindowIndices,
  numLocalObsWindows,
  numMaxPoints,
  numWindows,
  indicator
```

)

Arguments

x	Matrix containing the total number of detections (x[1,1]), the x- and y-coordinates (x[2:(x[1,1]+1),1:2]), and the corresponding detection window indices (x[2:(x[1,1]+1),3]) for a set of spatial points (detection locations).
lowerCoords, upperCoords	Matrices of lower and upper x- and y-coordinates of all detection windows scaled to the habitat (see (scaleCoordsToHabitatGrid). One row for each window. Each window should be of size 1x1.
s	Vector of x- and y-coordinates of the isotropic multivariate normal distribution mean (the AC location).
sd	Standard deviation of the isotropic multivariate normal distribution.
baseIntensities	Vector of baseline detection intensities for all detection windows.
habitatGridLocal	Matrix of rescaled habitat grid cells indices, from localIndices returned by the getLocalObjects function (
resizeFactor	Aggregation factor used in the getLocalObjects function to reduce the number of habitat grid cells.
localObsWindowIndices	Matrix of indices of local observation windows around each rescaled habitat grid cell, as returned by the getLocalObjects function (object named localIndices).
numLocalObsWindows	Vector of numbers of local observation windows around all habitat grid cells, as returned by the getLocalObjects function (object named numLocalIndices). The ith number gives the number of local (original) observation windows for the ith (rescaled) habitat window.
numMaxPoints	Maximum number of points. This value (non-negative integer) is only used when simulating detections to constrain the maximum number of detections.
numWindows	Number of detection windows. This value (positive integer) is used to truncate lowerCoords and upperCoords so that extra rows beyond numWindows are ignored.
indicator	Binary argument specifying whether the individual is available for detection (indicator = 1) or not (indicator = 0).
log	Logical argument, specifying whether to return the log-probability of the distribution.
n	Integer specifying the number of realisations to generate. Only n = 1 is supported.

Value

The (log) probability density of the observation matrix x.

Author(s)

Wei Zhang, Cyril Milleret and Pierre Dupont

References

W. Zhang, J. D. Chipperfield, J. B. Illian, P. Dupont, C. Milleret, P. de Valpine and R. Bischof. 2020. A hierarchical point process model for spatial capture-recapture data. bioRxiv. DOI 10.1101/2020.10.06.325035

C. Milleret, P. Dupont, C. Bonenfant, H. Brøseth, Ø. Flagstad, C. Sutherland and R. Bischof. 2019. A local evaluation of the individual state-space to scale up Bayesian spatial capture-recapture. Ecology and Evolution 9:352-363

#' @examples # Create habitat grid coordsHabitatGridCenter <- matrix(c(0.5, 3.5, 1.5, 3.5, 2.5, 3.5, 3.5, 3.5, 0.5, 2.5, 1.5, 2.5, 2.5, 2.5, 3.5, 2.5, 0.5, 1.5, 1.5, 1.5, 2.5, 1.5, 3.5, 1.5, 0.5, 0.5, 1.5, 0.5, 2.5, 0.5, 3.5, 0.5), ncol = 2, byrow = TRUE) colnames(coordsHabitatGridCenter) <- c("x","y") # Create observation windows lowerCoords <- matrix(c(1, 1, 2, 1, 1, 2, 2, 2), nrow = 4, byrow = TRUE) upperCoords <- matrix(c(2, 2, 3, 2, 2, 3, 3, 3), nrow = 4, byrow = TRUE) colnames(lowerCoords) <- colnames(upperCoords) <- c("x","y") # Plot check plot(coordsHabitatGridCenter[,"y"]~coordsHabitatGridCenter[,"x"],pch= points(lowerCoords[,"y"]~lowerCoords[,"x"],col="red",pch=16) points(upperCoords[,"y"]~upperCoords[,"x"],col="red",pch

s <- c(1, 1) sd <- 0.1 baseIntensities <- c(1:4) windowIndex <- 4 numPoints <- 1 numWindows <- 4 indicator <- 1

Rescale coordinates ScaledLowerCoords <- scaleCoordsToHabitatGrid(coordsData = lowerCoords, coordsHabitatGridCenter = coordsHabitatGridCenter)$coordsDataScaled ScaledUpperCoords <- scaleCoordsToHabitatGrid(coordsData = upperCoords, coordsHabitatGridCenter = coordsHabitatGridCenter)$coordsDataScaled ScaledUpperCoords[,2] <- ScaledUpperCoords[,2] + 1.5 ScaledLowerCoords[,2] <- ScaledLowerCoords[,2] - 1.5 habitatMask <- matrix(1, nrow = 4, ncol=4, byrow = TRUE) # Create local objects ObsWindowsLocal <- getLocalObjects(habitatMask = habitatMask, coords = ScaledLowerCoords, dmax=3, resizeFactor = 1, plot.check = TRUE)

Detection locations x <- matrix(c(1.5, 2, 1.1, 1.5, 1.4, 0.7, 2, 1.3, 1, 1.5), nrow = 5, byrow = TRUE)

get the window indeces on the third dimension of x windowIndexes <- 0 for(i in 1:nrow(x)) windowIndexes[i] <- getWindowIndex(curCoords = x[i,], lowerCoords = ScaledLowerCoords, upperCoords =ScaledUpperCoords)

x <- cbind(x, windowIndexes) # get the total number of detections on x[1,1] x <- rbind(c(length(windowIndexes),0,0), x) dpoisppLocalDetection_normal(x, ScaledLowerCoords, ScaledUpperCoords, s, sd, baseIntensities, ObsWindowsLocal$habitatGrid, ObsWindowsLocal$resizeFactor, ObsWindowsLocal$localIndices,ObsWindowsLocal numMaxPoints = dim(x)[1], numWindows, indicator, log = TRUE)

---

getHomeRangeArea *Computation of home range radius and area*

Description

`getHomeRangeArea` returns approximates estimates of home range radius and area for a given set of parameters with respect to a specified detection function using bisection algorithm. The following circular detection functions are available to use in nimbleSCR: half-normal (detFun = 0), half-normal plateau (detFun = 1), exponential (detFun = 2), asymmetric logistic (detFun = 3), bimodal (detFun = 4) and donut (detFun = 5).

Usage

```
getHomeRangeArea(
  x = 2,
  detFun = 0,
  prob = 0.95,
  d = 6,
  xlim = c(0, 30),
  ylim = c(0, 30),
  nBreaks = 800,
  tol = 0.00001,
  nIter = 2000
)
```

Arguments

x	Vector or matrix (parameters in columns) of values for different parameters corresponding to the specified detection function.
detFun	Numeric vNumeric variable denoting the type of detection function. 0 = Half-normal, 1 = Half-normal plateau, 2 = Exponential, 3 = Asymmetric logistic, 4 = Bimodal, 5 = Donut.
prob	Numeric variable denoting the quantile probability to compute the home range radius.
d	Numeric variable giving an initial value of the radius.
xlim	Vector of length 2 giving the range along x-axis.
ylim	Vector of length 2 giving the range along y-axis.
nBreaks	Numeric variable denoting the number of breaks along an axis.
tol	Numeric variable denoting the allowed tolerance in the radius estimate.
nIter	Numeric variable giving the maximum number of iterations in bisection algorithm.

Author(s)

Soumen Dey

References

Dey, S., Bischof, R., Dupont, P. P. A., & Milleret, C. (2022). Does the punishment fit the crime? Consequences and diagnosis of misspecified detection functions in Bayesian spatial capture–recapture modeling. Ecology and Evolution, 12, e8600. https://doi.org/10.1002/ece3.8600

Examples

```
## Not run:

# A user friendly vignette is also available on github:
# https://github.com/nimble-dev/nimbleSCR/blob/master/nimbleSCR/vignettes/
# Vignette name: Fit_with_dbinomLocal_normalPlateau_and_HomeRangeAreaComputation.rmd
```

```
# HALF-NORMAL PLATEAU FUNCTION (detFun = 1)
habitatMask <- matrix(1, nrow = 30, ncol= 30, byrow = TRUE)

prob <- 0.95
paramnames.hr <- c("HRradius", "HRarea")
sigma <- 1
w <- 1.5
params <- c(sigma, w)
names(params) <- c("sigma", "w")
HRAnim <- getHomeRangeArea( x = params, detFun = 1, prob = prob, d = 6,
                     xlim = c(0, dim(habitatMask)[2]), ylim = c(0, dim(habitatMask)[1]),
                      nBreaks = 800, tol = 1E-5, nIter = 2000)

# Different values of argument "detFun"
# 0 = Half-normal, 1 = Half-normal plateau, 2 = Exponential,
# 3 = Aysmmetric logistic, 4 = Bimodal, 5 = Donut.
HR.hnp <- c(HRAnim$run())
names(HR.hnp) <- paramnames.hr
print(HR.hnp)
# FASTER HRA COMPUTATION USING NIMBLE
samples <- cbind(rgamma(n = 500, shape = 1, rate = 1), rgamma(n = 500, shape = 1.5, rate = 1))
colnames(samples) <- c("sigma", "w")
HRAnim.mat <- getHomeRangeArea(x = samples, detFun = 1, prob = prob, d = 6,
                      xlim = c(0, dim(habitatMask)[2]), ylim = c(0, dim(habitatMask)[1]),
                         nBreaks = 800, tol = 1E-5, nIter = 2000)

cHRAnim.arr <- compileNimble(HRAnim.mat, resetFunctions = TRUE)

HRA.Runtime <- system.time(
  HR.chain <- cHRAnim.arr$run()
)
print(HRA.Runtime)
dimnames(HR.chain)[[2]] <- paramnames.hr
HRest <- do.call(rbind, lapply(c(1:2), function(j){
  c(mean(HR.chain[,j], na.rm = TRUE), sd(HR.chain[,j], na.rm = TRUE))
}))
dimnames(HRest) <- list(paramnames.hr, c("MEAN", "SD"))

cat("Numerical estimates using MCMC samples: \n", sep = "")
print(HRest)

# HALF-NORMAL FUNCTION (detFun = 0)
sigma = 2
params <- c(sigma)
names(params) <- c("sigma")

HRAnim <- getHomeRangeArea(x = params, detFun = 0, prob = prob, d = 6,
                     xlim = c(0, dim(habitatMask)[2]), ylim = c(0, dim(habitatMask)[1]),
                     nBreaks = 800, tol = 1E-5, nIter = 2000)

HR.hn <- c(HRAnim$run())
names(HR.hn) <- paramnames.hr
```

```
print(HR.hn)

# Exponential (detFun = 2)

rate = 1/2
params <- c(rate)
names(params) <- c("rate")
HRAnim <- getHomeRangeArea(x = params, detFun = 2, prob = prob, d = 6,
                     xlim = c(0, dim(habitatMask)[2]), ylim = c(0, dim(habitatMask)[1]),
                     nBreaks = 800, tol = 1E-5, nIter = 2000)
HR.exp <- c(HRAnim$run())
names(HR.exp) <- paramnames.hr
print(HR.exp)

# Asymmetric logistic (detFun = 3)

sigma = 2
alpha.a = 5
alpha.b = 1
params <- c(sigma, alpha.a, alpha.b)
names(params) <- c("sigma", "alpha.a", "alpha.b")
HRAnim <- getHomeRangeArea(x = params, detFun = 3, prob = prob, d = 6,
                     xlim = c(0, dim(habitatMask)[2]), ylim = c(0, dim(habitatMask)[1]),
                     nBreaks = 800, tol = 1E-5, nIter = 2000)
HR.al <- c(HRAnim$run())
names(HR.al) <- paramnames.hr
print(HR.al)


# Bimodal (detFun = 4)

p0.a = 0.25
sigma.a = 0.5
p0.b = 0.15
sigma.b = 1
w = 2
params <- c(sigma.a, sigma.b, p0.a, p0.b, w)
names(params) <- c("sigma.a", "sigma.b", "p0.a", "p0.b", "w")
HRAnim <- getHomeRangeArea(x = params, detFun = 4, prob = prob, d = 6,
                     xlim = c(0, dim(habitatMask)[2]), ylim = c(0, dim(habitatMask)[1]),
                     nBreaks = 800, tol = 1E-5, nIter = 2000)
HR.bi <- c(HRAnim$run())
names(HR.bi) <- paramnames.hr
print(HR.bi)

# Donut (detFun = 5)

sigma.a = 1.5
sigma.b = 1
w = 1
params <- c(sigma.a, sigma.b, w)
names(params) <- c("sigma.a", "sigma.b", "w")
HRAnim <- getHomeRangeArea(x = params, detFun = 5, prob = prob, d = 6,
```

```
                    xlim = c(0, dim(habitatMask)[2]), ylim = c(0, dim(habitatMask)[1]),
                    nBreaks = 800, tol = 1E-5, nIter = 2000)
HR.dn <- c(HRAnim$run())
names(HR.dn) <- paramnames.hr
print(HR.dn)


## End(Not run)
```

`getLocalObjects` *Local Objects Identification*

Description

R utility function to identify all objects (e.g. traps) within a given radius dmax of each cell in a habitat mask. Used in the implementation of the local evaluation approach in SCR models (`dbinomLocal_normal`;`dpoisLocal_normal`). The distance to the activity center and the detection probability are then calculated for local objects only (i.e. the detection probability is assumed to be 0 for all other objects as they are far enough from the activity center).

Usage

```
getLocalObjects(habitatMask, coords, dmax, resizeFactor = 1, plot.check = TRUE)
```

Arguments

`habitatMask`	a binary matrix object indicating which cells are considered as suitable habitat.
`coords`	A matrix giving the x- and y-coordinate of each object (i.e. trap). x- and y-coordinates should be scaled to the habitat (`scaleCoordsToHabitatGrid`).
`dmax`	The maximal radius from a habitat cell center within which detection probability is evaluated locally for each trap.
`resizeFactor`	An aggregation factor to reduce the number of habitat cells to retrieve local objects for. Defaults to 1; no aggregation.
`plot.check`	A visualization option (if TRUE); displays which objects are considered "local objects" for a randomly chosen habitat cell.

Details

The `getLocalObjects` function is used in advance of model building.

Value

This function returns a list of objects:

- localIndices: a matrix with number of rows equal to the reduced number of habitat grid cells (following aggregation). Each row gives the id numbers of the local objects associated with this grid cell.

- habitatGrid: a matrix of habitat grid cells ID corresponding to the row indices in localIndices.
- numLocalIndices: a vector of the number of local objects for each habitat grid cell in habitatGrid.
- numLocalIndicesMax: the maximum number of local objects for any habitat grid cell ; corresponds to the number of columns in habitatGrid.
- resizeFactor: the aggregation factor used to reduce the number of habitat grid cells.

Author(s)

Cyril Milleret and Pierre Dupont

Examples

```
colNum <- sample(20:100,1)
rowNum <- sample(20:100,1)
coords <- expand.grid(list(x = seq(0.5, colNum, 1),
                           y = seq(0.5, rowNum, 1)))

habitatMask <- matrix(rbinom(colNum*rowNum, 1, 0.8), ncol = colNum, nrow = rowNum)

localObject.list <- getLocalObjects(habitatMask, coords,  dmax = 7,resizeFactor = 1)
```

---

getMidPointNodes *Generate midpoint integration nodes*

Description

Generate midpoint nodes and weights for integrating a function numerically over a set of windows. For each window, generate a set of equally spaced nodes and weights.

Usage

```
getMidPointNodes(lowerCoords, upperCoords, numSubintervals = 10)
```

Arguments

`lowerCoords`, `upperCoords`
: Matrices of lower and upper x- and y-coordinates of a set of windows. One row for each window.

`numSubintervals`
: Number of subintervals each dimension of a window is divided into.

Value

A list of midpoint nodes and weights.

Author(s)

Wei Zhang

Examples

```
lowerCoords <- matrix(c(0, 0, 1, 0, 0, 1, 1, 1), nrow = 4, byrow = TRUE)
upperCoords <- matrix(c(1, 1, 2, 1, 1, 2, 2, 2), nrow = 4, byrow = TRUE)
getMidPointNodes(lowerCoords, upperCoords, 5)
```

---

getSparseY *Sparse Matrix Preparation*

Description

R utility function to turn a two or three-dimensional detection array into a sparse matrix representation (see Turek et al., 2021 <doi.org/10.1002/ecs2.3385> for more details). Used in the implementation of the dbinomLocal_normal and dpoisLocal_normal functions.

Usage

```
getSparseY(x, noDetections = -1, nMaxTraps = NULL)
```

Arguments

x	A two- or three-dimensional observation data array with dimensions : number of individuals, number of traps, (and number of detection occasions/sessions).
noDetections	The value indicating no detection. Defaults to -1.
nMaxTraps	The maximum number of traps at which detections can occur. It is necessary to artificially augment the sparse detection array when using the random generation functionality of the rbinomLocal_normal or rpoisLocal_normal functions. When simulating detection data, augmenting the size of the detection array is necessary to avoids artificially limiting the number of detectors at which individuals can be detected. Default value is maxDetNums * 2, which doubles the maximum number of traps at which an individual can be detected. We generally recommend using *numLocalIndicesMax* obtained from getLocalObjects when aiming at randomly generating detections from rbinomLocal_normal or rpoisLocal_normal.

Details

The getSparseY function is used in advance of model building to create a sparse matrix representation of the observation data. It creates and returns a list of objects:

Value

A list of objects which constitute a sparse representation of the observation data:

- *detNums* A matrix with number of traps at which each individual (in rows) was detected at each occasions/sessions (in columns).
- *maxDetNums* The maximum number of traps at which an individual was detected (i.e., the maximum of *detNums*).
- *detIndices* An array of dimensions n.individuals, maxDetNums, and number of occasions/sessions, which contains the IDs of the traps where each individual was detected.
- *y* An array of dimensions n.individuals, maxDetNums, and occasions/sessions, which contains the number of observations of each individual at the traps it was detected at.
- *yCombined* An array that combines *detNums*, *y*, and *detIndices* by columns (in that specific order). Note that *y*, and *detIndices* are augmented before combining, such that the maximum number of detectors at which an individual can be detected is equal to *nMaxTraps* Consequently, the number of columns of *lengthYCombined* is 2*nMaxTraps + 1.
- *lengthYCombined* Dimension of the augmented lengthYCombined object to be specified as the argument *lengthYCombined* of the `dbinomLocal_normal` or `dpoisLocal_normal` functions when simulating detection data.

Author(s)

Cyril Milleret

Examples

```
y.full <- matrix(rbinom(5000, 5, 0.02), ncol = 100)

y <- getSparseY(y.full)
```

---

getWindowCoords *Get lower and upper windows coordinates*

Description

The `getWindowCoords` is an R utility function to create lower and upper habitat and observation windows coordinates, as well an habitat grid with cell ids. Those objects are necessary to run all point process (pp) functions. All input data should be scaled to the habitat grid using `scaleCoordsToHabitatGrid`. Note that we assume homogeneous window sizes.

Usage

```
getWindowCoords(
  scaledHabGridCenter = scaledHabGridCenter,
  scaledObsGridCenter = NULL,
  plot.check = TRUE
)
```

Arguments

scaledHabGridCenter	A matrix with the scaled "x" and "y" habitat windows grid cell centers (after using scaleCoordsToHabitatGrid).
scaledObsGridCenter	A matrix with the scaled "x" and "y" observation windows grid cell centers (afer using scaleCoordsToHabitatGrid). This is an optional argument and only necessary when modelling detection as a point process (e.g. dpoisppDetection_normal).
plot.check	A visualization option (if TRUE); displays habitat and detection windows.

Value

A list of objects :

- *lowerHabCoords* A matrix with the "x" and "y" lower habitat window coordinates.
- *upperHabCoords* A matrix with the "x" and "y" upper habitat window coordinates.
- *habitatGrid* A matrix of habitat cell ID that can be used to lookup efficiently the cell ID from a coordinate scaled to the habitat grid: habitatGrid[trunc(scaledHabGridCenter[1,"y"]) + 1, trunc(scaledHabGridCenter[1,"x"]) + 1]. See scaleCoordsToHabitatGrid for more details.
- *lowerObsCoords* A matrix with the "x" and "y" lower observation window coordinates. Only returned when *scaledObsGridCenter* is provided.
- *upperObsCoords* A matrix with the "x" and "y" upper observation window coordinates. Only returned when *scaledObsGridCenter* is provided.

Author(s)

Cyril Milleret

Examples

```
coordsGridCenter <- expand.grid(list(x = seq(50.5, 100, 1),
                                     y = seq(100.5, 150, 1)))
coordsData <- expand.grid(list(x = seq(60, 90, 1),
                               y = seq(110, 140, 1)))

plot(coordsGridCenter[,2] ~ coordsGridCenter[,1])
points(coordsData[,2] ~ coordsData[,1], col="red")
scaled <- scaleCoordsToHabitatGrid(coordsData = coordsData
                                   , coordsHabitatGridCenter = coordsGridCenter)
plot(scaled$coordsHabitatGridCenterScaled[,2] ~ scaled$coordsHabitatGridCenterScaled[,1])
points(scaled$coordsDataScaled[,2] ~ scaled$coordsDataScaled[,1], col="red")

LowerAndUpperCoords <- getWindowCoords(scaledHabGridCenter = scaled$coordsHabitatGridCenterScaled,
                                       scaledObsGridCenter = scaled$coordsDataScaled)

# Plot habitat window cell centers and lower/upper coordinates
plot(scaled$coordsHabitatGridCenterScaled[,2] ~
     scaled$coordsHabitatGridCenterScaled[,1],
     pch=16, cex=0.3, col=grey(0.5))
```

```
points(LowerAndUpperCoords$lowerHabCoords[,2] ~
       LowerAndUpperCoords$lowerHabCoords[,1],
       pch=16, cex=0.3, col=grey(0.1))
points(LowerAndUpperCoords$upperHabCoords[,2] ~
       LowerAndUpperCoords$upperHabCoords[,1],
       pch=16, cex=0.3, col=grey(0.1))

# Plot observation window cells center and lower/upper coordinates
points(scaled$coordsDataScaled[,2]~scaled$coordsDataScaled[,1], pch=16,
 cex=0.3, col = adjustcolor("red",alpha.f = 0.8))
points(LowerAndUpperCoords$lowerObsCoords[,2] ~
        LowerAndUpperCoords$lowerObsCoords[,1],
       pch=16, cex=0.3, col = adjustcolor("red", alpha.f = 0.8))
points(LowerAndUpperCoords$upperObsCoords[,2] ~
       LowerAndUpperCoords$upperObsCoords[,1],
       pch=16, cex=0.3, col = adjustcolor("red", alpha.f = 0.8))
```

`getWindowIndex` *Get window index*

Description

From a set of windows, find the index of the window into which a given point falls. Can be applied to detection and habitat windows.

Usage

```
getWindowIndex(curCoords, lowerCoords, upperCoords)
```

Arguments

`curCoords`	Vector of coordinates of a single spatial point
`lowerCoords`, `upperCoords`	Matrices of lower and upper x- and y-coordinates of a set of windows. One row for each window.

Value

Index of the window where the given point falls; -1 is returned if the point does not fall in any window.

Author(s)

Pierre Dupont

Examples

```
sourceCoords <- c(1.5,2.2)
lowerCoords <- cbind(c(0,1,3,0),c(0,1,2,2))
upperCoords <- cbind(c(1,3,5,3),c(1,2,4,4))
getWindowIndex(sourceCoords, lowerCoords, upperCoords)
```

---

integrateIntensityLocal_normal

Integrate the multivariate normal intensity with local evaluation

---

Description

Calculate the integral of the intensity function with an isotropic multivariate normal kernel over a set of windows. The local evaluation technique is implemented.

Usage

```
integrateIntensityLocal_normal(
  lowerCoords,
  upperCoords,
  s,
  baseIntensities,
  sd,
  numLocalWindows,
  localWindows
)
```

Arguments

`lowerCoords`, `upperCoords`	Matrices of lower and upper x- and y-coordinates of a set of windows. One row for each window.
`s`	Vector of x- and y-coordinates of the isotropic multivariate normal distribution mean (AC location).
`baseIntensities`	Vector of baseline intensities for all windows.
`sd`	Standard deviation of the isotropic multivariate normal distribution.
`numLocalWindows`	Number of windows that are close to the activity center
`localWindows`	Vector of indices of the windows that are close to the activity center.

Value

A vector of integrated intensities over all local windows.

Author(s)

Cyril Milleret and Wei Zhang

References

W. Zhang, J. D. Chipperfield, J. B. Illian, P. Dupont, C. Milleret, P. de Valpine and R. Bischof. 2020.
A hierarchical point process model for spatial capture-recapture data. bioRxiv. DOI 10.1101/2020.10.06.325035

Examples

```
lowerCoords <- matrix(c(0, 0, 1, 0, 0, 1, 1, 1), nrow = 4, byrow = TRUE)
upperCoords <- matrix(c(1, 1, 2, 1, 1, 2, 2, 2), nrow = 4, byrow = TRUE)
s <- c(0.1, 0.9)
sd <- 0.1
baseIntensities <- c(1:4)
numLocalWindows <- 2
localWindows <- c(1, 3)
integrateIntensityLocal_normal(lowerCoords, upperCoords, s,
                               baseIntensities, sd,
                               numLocalWindows, localWindows)
```

---

integrateIntensity_exp

Integrate the multivariate exponential intensity

Description

Calculate the integral of the intensity function with an isotropic multivariate exponential kernel over a set of windows.

Usage

```
integrateIntensity_exp(
  lowerCoords,
  upperCoords,
  s,
  baseIntensities,
  lambda,
  numWindows
)
```

Arguments

`lowerCoords`, `upperCoords`	Matrices of lower and upper x- and y-coordinates of a set of windows. One row for each window.
`s`	Vector of x- and y-coordinates of the AC location.

baseIntensities
: Vector of baseline intensities for all windows.

lambda
: Rate parameter of the isotropic multivariate exponential distribution.

numWindows
: Total number of windows. This value (positive integer) is used to truncate lowerCoords and upperCoords so that extra rows beyond numWindows are ignored.

Value

A vector of integrated intensities over all windows.

Author(s)

Wei Zhang

References

W. Zhang, J. D. Chipperfield, J. B. Illian, P. Dupont, C. Milleret, P. de Valpine and R. Bischof. 2020.
A hierarchical point process model for spatial capture-recapture data. bioRxiv. DOI 10.1101/2020.10.06.325035

Examples

```
lowerCoords <- matrix(c(0, 0, 1, 0, 0, 1, 1, 1), nrow = 4, byrow = TRUE)
upperCoords <- matrix(c(1, 1, 2, 1, 1, 2, 2, 2), nrow = 4, byrow = TRUE)
s <- c(1, 1)
lambda <- 1.0
baseIntensities <- c(1:4)
numWindows <- 4
integrateIntensity_exp(lowerCoords, upperCoords, s, baseIntensities, lambda, numWindows)
```

---

integrateIntensity_normal

Integrate the multivariate normal intensity

Description

Calculate the integral of the intensity function with an isotropic multivariate normal kernel over a set of windows.

Usage

```
integrateIntensity_normal(
  lowerCoords,
  upperCoords,
  s,
  baseIntensities,
  sd,
  numWindows
)
```

Arguments

lowerCoords, upperCoords	Matrices of lower and upper x- and y-coordinates of a set of windows. One row for each window.
s	Vector of x- and y-coordinates of the isotropic multivariate normal distribution mean (AC location).
baseIntensities	Vector of baseline intensities for all windows.
sd	Standard deviation of the isotropic multivariate normal distribution.
numWindows	Total number of windows. This value (positive integer) is used to truncate lowerCoords and upperCoords so that extra rows beyond numWindows are ignored.

Value

A vector of integrated intensities over all windows.

Author(s)

Wei Zhang

References

W. Zhang, J. D. Chipperfield, J. B. Illian, P. Dupont, C. Milleret, P. de Valpine and R. Bischof. 2020.
A hierarchical point process model for spatial capture-recapture data. bioRxiv. DOI 10.1101/2020.10.06.325035

Examples

```
lowerCoords <- matrix(c(0, 0, 1, 0, 0, 1, 1, 1), nrow = 4, byrow = TRUE)
upperCoords <- matrix(c(1, 1, 2, 1, 1, 2, 2, 2), nrow = 4, byrow = TRUE)
s <- c(1, 1)
sd <- 0.1
baseIntensities <- c(1:4)
numWindows <- 4
integrateIntensity_normal(lowerCoords, upperCoords, s, baseIntensities, sd, numWindows)
```

---

localTrapCalculations *Local Trap Calculations*

Description

These functions are deprecated, and they will be removed from a future release. Utility functions to enable local trap calculations in SCR models. See details section for more information.

Usage

```
makeGrid(xmin = 0, ymin = 0, xmax, ymax, resolution = 1, buffer = 0)

findLocalTraps(grid, trapCoords, dmax)

getNumLocalTraps(idarg, nLocalTraps, LTD1arg)

getLocalTrapIndices(MAXNUM, localTraps, n, idarg)

calcLocalTrapDists(MAXNUM, n, localTrapInd, s, trapCoords)

calcLocalTrapExposure(R, n, d, localTrapInd, sigma, p0)
```

Arguments

xmin	Minimal value among all trap location x-coordinates.
ymin	Minimal value among all trap location y-coordinates.
xmax	Maximal value among all trap location x-coordinates.
ymax	Maximal value among all trap location y-coordinates.
resolution	Desired resolution (in both x and y directions) of discretized grid.
buffer	Horizontal and vertical buffer for discretized grid, specifying how much it should extend (above, below, left, and right) of the maximal trap locations.
grid	The grid object returned from the makeGrid function.
trapCoords	An nTraps x 2 array giving giving the x- and y-coordinate locations of all traps.
dmax	The maximal radius from an activity center for performing trap calculations (dmax).
idarg	A grid id, returned from the makeID function inside model code.
nLocalTraps	The number of local traps to all grid cells, which is given by the first column of the localTraps array.
LTD1arg	The number of columns in the localTraps array.
MAXNUM	The maximum number of local traps among all grid cells. This is given by the (number of rows)-1 of the localTraps array.
localTraps	The array returned from the findLocalTraps function.
n	The number of local traps to a specified grid cell, as return.
localTrapInd	The indices of the local traps to a grid cell, as returned by the getLocalTrapIndices function.
s	A length-2 vector giving the activity center of an indiviual.
R	The total number of traps.
d	A vector of distances from an activity center to the local traps.
sigma	Scale of decay for detection probability.
p0	Baseline detection probability.

Details

These functions are deprecated, and they will be removed from a future release.

The makeGrid function is used in advance of model building. It creates and returns a list of two objects: a table (grid) corresponding to the discretized grid, where each row gives the x-coordinate, the y-coordinate, and the id number for a grid cell; and second, a function (makeID) to be used in the model code which operates on a discretized AC location, and returns the id number of the corresponding grid cell.

The findLocalTraps function operates on the grid object returned from makeGrid, and an array of the trap location coordinates, and the desired maximal exposure radius for caluclations (dmax). It returns a array (localTraps) with number of rows equal to the number of grid cells. The first element of each row gives the number of local traps within exposure radius to that grid cell. The following elements of each row give the id numbers of those local traps.

A visualization function (plotTraps) is also provided in the example code, which displaces the discretized grid (small black points), all trap locations (green circles), a specified grid cell location (specified by i) as a large X, and the local traps to that specified grid cell (red circles).

The getNumLocalTraps function is used inside the model code. It operates on an id for a grid cell, the localTraps array (generated by findLocalTraps), and the constant value LTD1. This function returns the number of traps which are local to a specified grid cell.

The getLocalTrapIndices function is used inside the model code. It returns a vector containing the ids of the local traps to a particular grid cell.

The calcLocalTrapDists function is used inside the model code. It calculates the distances from an activity center, to the local traps relative to the grid cell nearest that activity center.

The calcLocalTrapExposure function is specific to the detection probability calculations used in this example. This function should be modified specifically to the detection function, exposure function, or otherwise calculations to be done only for the traps in the vicinity of individual activity center locations

Author(s)

Daniel Turek

Examples

```
## Not run:

## generate random trap locations
nTraps <- 200
traps_xmin <- 0
traps_ymin <- 0
traps_xmax <- 100
traps_ymax <- 200
set.seed(0)
traps_xCoords <- round(runif(nTraps, traps_xmin, traps_xmax))
traps_yCoords <- round(runif(nTraps, traps_ymin, traps_ymax))
trap_coords <- cbind(traps_xCoords, traps_yCoords)

## buffer distance surrounding sides of rectangular discretization grid
```

```
## which overlays trap locations
buffer <- 10

## resolution of rectangular discretization grid
resolution <- 10

## creates grid and makeID function,
## for grid overlaying trap locations,
## and to lookup nearest grid cell to any AC
makeGridReturn <- makeGrid(xmin = traps_xmin, xmax = traps_xmax,
                           ymin = traps_ymin, ymax = traps_ymax,
                           buffer = buffer,
                           resolution = resolution)

grid <- makeGridReturn$grid
makeID <- makeGridReturn$makeID

## maximum radis within an individual AC to perform trap calculations,
dmax <- 30

## n = localTraps[i,1] gives the number of local traps
## localTraps[i, 2:(n+1)] gives the indices of the local traps
localTraps <- findLocalTraps(grid, trap_coords, dmax)

plotTraps <- function(i, grid, trap_coords, localTraps) {
    plot(grid[,1], grid[,2], pch = '.', cex=2)
    points(trap_coords[,1], trap_coords[,2], pch=20, col='forestgreen', cex=1)
    if(!missing(i)) {
        i <- max(i %% dim(grid)[1], 1)
        n <- localTraps[i,1]
        trapInd <- numeric(0)
        if(n > 0)  trapInd <- localTraps[i,2:(n+1)]
        theseTraps <- trap_coords[trapInd,, drop = FALSE]
        points(theseTraps[,1], theseTraps[,2], pch = 20, col = 'red', cex=1.5)
        points(grid[i,1], grid[i,2], pch = 'x', col = 'blue', cex=3)
    }
}

## visualise some local traps
plotTraps(10,  grid, trap_coords, localTraps)
plotTraps(200, grid, trap_coords, localTraps)
plotTraps(380, grid, trap_coords, localTraps)

## example model code
## using local trap calculations
code <- nimbleCode({
    sigma ~ dunif(0, 100)
    p0 ~ dunif(0, 1)
    for(i in 1:N) {
        S[i,1] ~ dunif(0, xmax)
        S[i,2] ~ dunif(0, ymax)
        Sdiscrete[i,1] <- round(S[i,1]/res) * res
        Sdiscrete[i,2] <- round(S[i,2]/res) * res
```

```
        id[i] <- makeID( Sdiscrete[i,1:2] )
        nLocalTraps[i] <- getNumLocalTraps(id[i], localTraps[1:LTD1,1], LTD1)
        localTrapIndices[i,1:maxTraps] <-
          getLocalTrapIndices(maxTraps, localTraps[1:LTD1,1:LTD2], nLocalTraps[i], id[i])
        d[i, 1:maxTraps] <- calcLocalTrapDists(
            maxTraps, nLocalTraps[i], localTrapIndices[i,1:maxTraps],
            S[i,1:2], trap_coords[1:nTraps,1:2])
        g[i, 1:nTraps] <- calcLocalTrapExposure(
        nTraps, nLocalTraps[i], d[i,1:maxTraps], localTrapIndices[i,1:maxTraps], sigma, p0)
        y[i, 1:nTraps] ~ dbinom_vector(prob = g[i,1:nTraps], size = trials[1:nTraps])
    }
})

## generate random detection data; completely random
N <- 100
set.seed(0)
y <- array(rbinom(N*nTraps, size=1, prob=0.8), c(N, nTraps))

## generate AC location initial values
Sinit <- cbind(runif(N, traps_xmin, traps_xmax),
               runif(N, traps_ymin, traps_ymax))

constants <- list(N = N,
                  nTraps = nTraps,
                  trap_coords = trap_coords,
                  xmax = traps_xmax,
                  ymax = traps_ymax,
                  res = resolution,
                  localTraps = localTraps,
                  LTD1 = dim(localTraps)[1],
                  LTD2 = dim(localTraps)[2],
                  maxTraps = dim(localTraps)[2] - 1)

data <- list(y = y, trials = rep(1,nTraps))

inits <- list(sigma = 1,
              p0 = 0.5,
              S = Sinit)

## create NIMBLE model object
Rmodel <- nimbleModel(code, constants, data, inits,
                      calculate = FALSE, check = FALSE)

## use model object for MCMC, etc.


## End(Not run)
```

`marginalVoidProbIntegrand` *Integrand of the marginal void probability integral*

Description

Integrand of the marginal void probability integral. The domain of this function is the habitat domain.

Usage

```
marginalVoidProbIntegrand(
  x,
  lowerCoords,
  upperCoords,
  sd,
  baseIntensities,
  numPoints,
  numWindows
)
```

Arguments

`x`	Matrix of x- and y-coordinates of a set of spatial points. One row corresponds to one point.
`lowerCoords`, `upperCoords`	Matrices of lower and upper x- and y-coordinates of a set of detection windows. One row for each window.
`sd`	Standard deviation of the isotropic multivariate normal distribution.
`baseIntensities`	Vector of baseline detection intensities for all detection windows.
`numPoints`	Number of points that should be considered. This value (positive integer) is used to truncate `x` so that extra rows beyond `numPoints` are ignored.
`numWindows`	Number of windows. This value (positive integer) is used to truncate `lowerCoords` and `upperCoords` so that extra rows beyond `numWindows` are ignored.

Value

A vector of values of the integrand evaluated at each point of `x`.

Author(s)

Wei Zhang

References

W. Zhang, J. D. Chipperfield, J. B. Illian, P. Dupont, C. Milleret, P. de Valpine and R. Bischof. 2020.
A hierarchical point process model for spatial capture-recapture data. bioRxiv. DOI 10.1101/2020.10.06.325035

Examples

```
x <- matrix(runif(10, 0, 2), nrow = 5)
lowerCoords <- matrix(c(0, 0, 1, 0, 0, 1, 1, 1), nrow = 4, byrow = TRUE)
upperCoords <- matrix(c(1, 1, 2, 1, 1, 2, 2, 2), nrow = 4, byrow = TRUE)
sd <- 0.1
baseIntensities <- c(1:4)
numPoints <- 5
numWindows <- 4
marginalVoidProbIntegrand(x, lowerCoords, upperCoords, sd, baseIntensities, numPoints, numWindows)
```

---

marginalVoidProbNumIntegration

Marginal void probability

Description

Calculate the marginal void probability using the midpoint integration method.

Usage

```
marginalVoidProbNumIntegration(
  quadNodes,
  quadWeights,
  numNodes,
  lowerCoords,
  upperCoords,
  sd,
  baseIntensities,
  habIntensities,
  sumHabIntensity,
  numObsWindows,
  numHabWindows
)
```

Arguments

quadNodes	Three-dimensional array of nodes for midpoint integration. The dimension sizes are equal to the number of nodes per habitat window (1st), 2 (2nd), and the number of habitat windows (3rd).
quadWeights	Vector of weights for midpoint integration.
numNodes	Vector of numbers of nodes for all habitat windows.
lowerCoords, upperCoords	Matrix of lower and upper x- and y-coordinates of all detection windows. One row for each window.
sd	Standard deviation of the isotropic multivariate normal distribution.

`baseIntensities`
: Vector of baseline detection intensities for all detection windows.

`habIntensities`
: Vector of habitat intensities for all habitat windows.

`sumHabIntensity`
: Total habitat selection intensity over all windows.

`numObsWindows`
: Number of detection windows.

`numHabWindows`
: Number of habitat windows.

Value

The marginal void probability.

Author(s)

Wei Zhang

References

W. Zhang, J. D. Chipperfield, J. B. Illian, P. Dupont, C. Milleret, P. de Valpine and R. Bischof. 2020.
A hierarchical point process model for spatial capture-recapture data. bioRxiv. DOI 10.1101/2020.10.06.325035

Examples

```
lowerHabCoords <- matrix(c(0, 0, 0, 1), nrow = 2, byrow = TRUE)
upperHabCoords <- matrix(c(2, 1, 2, 2), nrow = 2, byrow = TRUE)
lowerObsCoords <- matrix(c(0, 0, 1, 0, 0, 1, 1, 1), nrow = 4, byrow = TRUE)
upperObsCoords <- matrix(c(1, 1, 2, 1, 1, 2, 2, 2), nrow = 4, byrow = TRUE)
nodesRes <- getMidPointNodes(lowerHabCoords, upperHabCoords, 10)
quadNodes <- nodesRes$quadNodes
quadWeights <- nodesRes$quadWeights
numNodes <- rep(100, 2)
sd <- 0.1
baseDetIntensities <- c(1:4)
habIntensities <- c(1:2)
sumHabIntensity <- sum(habIntensities * c(2, 2))
numObsWindows <- 4
numHabWindows <- 2
marginalVoidProbNumIntegration(quadNodes, quadWeights, numNodes,
                               lowerObsCoords, upperObsCoords, sd,
                               baseDetIntensities, habIntensities,
                               sumHabIntensity, numObsWindows, numHabWindows)
```

sampler_categorical_general
nimble *MCMC sampler function for general categorial distributions*

Description

The categorical_general sampler operates within nimble's MCMC engine to perform Gibbs sampling for a single node, which must in essence follow a categorical distribution. However, the prior distribution need not be nimble's dcat distribution, but rather can be any (potentially user-defined) distribution which has the same support as a standard categorical (dcat) distribution. Specifically: the distribution must define a discrete random variable, which can only attain values from the set 1, 2, 3, ..., numCategories.

The categorical_general sampler requires one control list argument, named numCategories, which specifies the fixed upper-bound for the range of the random variable.

The categorical_general sampler is designed to be used in nimble's MCMC engine, and can be added to an MCMC configuration object using the addSampler method. See help(configureMCMC) for more information about MCMC configuration objects and adding custom samplers.

Usage

```
sampler_categorical_general(model, mvSaved, target, control)
```

Arguments

model	(uncompiled) model on which the MCMC is to be run.
mvSaved	modelValues object to be used to store MCMC samples.
target	node on which the sampler will operate.
control	named list containing an elemente named numCategories, which specifies the upper-bound for the range of the random variable.

Author(s)

Daniel Turek

Examples

```
## Not run:
## define custom dmy_categorical distribution as a nimbleFunction
dmy_categorical <- nimbleFunction(...)

## nimble model code, using custom-written dmy_categorical distribution
code <- nimbleCode({
  x ~ dmy_categorical(...)
})

## create NIMBLE model object
Rmodel <- nimbleModel(code)
```

```
## create MCMC configuration object with no samplers
conf <- configureMCMC(Rmodel, nodes = NULL)

## add categorical_general sampler to MCMC configuration
conf$addSampler(target = 'x', type = 'categorical_general', control = list(numCategories = 10))

## build MCMC algorithm
Rmcmc <- buildMCMC(conf)

## compile model and MCMC, run MCMC algorithm

## End(Not run)
```

`scaleCoordsToHabitatGrid` *Scale x- and y-coordinates to grid cells coordinates.*

Description

R utility function to scale x- and y- coordinates to the habitat grid. Scaling the coordinates to the habitat grid allows implementation of the fast look-up approach to identify the habitat grid cell in which a point is located. This technique was first applied by Mike Meredith in SCR (https://mmeredith.net/blog/2013/1309_SECR_in_JAGS_patchy_habitat.htm). Re-scaling the entire coordinate system of the data input is a requirement to run SCR models with the local evaluation approach. This function requires square grid cells and coordinates using projection with units in meters or km (e.g., UTM but not latitude/longitude)

Usage

```
scaleCoordsToHabitatGrid(
  coordsData = coordsData,
  coordsHabitatGridCenter = coordsHabitatGridCenter,
  scaleToGrid = TRUE
)
```

Arguments

`coordsData`	A matrix or array of x- and y-coordinates to be scaled to the habitat grid. x- and y- coordinates must be identified using "x" and "y" dimnames.
`coordsHabitatGridCenter`	A matrix of x- and y-coordinates for each habitat grid cell center.
`scaleToGrid`	Defaults to TRUE. If FALSE, coordsData are already scaled and will be rescaled to its original coordinates.

Value

This function returns a list of objects:

- coordsDataScaled: A matrix or array of scaled (rescaled if scaleToGrid==FALSE) x- and y-coordinates for coordsData.
- coordsHabitatGridCenterScaled: A matrix of scaled x- and y-cell coordinates for coordsHabitatGridCenter.

Author(s)

Richard Bischof, Cyril Milleret

Examples

```
coordsGridCenter <- expand.grid(list(x = seq(50.5, 100, 1),
                                     y = seq(100.5, 150, 1)))
coordsData <- expand.grid(list(x = seq(60, 90, 1),
                               y = seq(110, 140, 1)))
plot(coordsGridCenter[,2]~coordsGridCenter[,1])
points(coordsData[,2]~coordsData[,1], col="red")
scaled <- scaleCoordsToHabitatGrid(coordsData = coordsData
                                   , coordsHabitatGridCenter = coordsGridCenter)
plot(scaled$coordsHabitatGridCenterScaled[,2]~scaled$coordsHabitatGridCenterScaled[,1])
points(scaled$coordsDataScaled[,2]~scaled$coordsDataScaled[,1], col="red")
```

---

stratRejectionSampler_exp

Stratified rejection sampler for multivariate exponential point process

---

Description

Simulate data using a stratified rejection sampler from a point process with an isotropic multivariate exponential decay kernel.

Usage

```
stratRejectionSampler_exp(
  numPoints,
  lowerCoords,
  upperCoords,
  s,
  windowIntensities,
  lambda
)
```

Arguments

numPoints	Number of spatial points to generate.
lowerCoords, upperCoords	Matrices of lower and upper x- and y-coordinates of a set of detection windows. One row for each window.
s	Vector of x- and y-coordinates of of the isotropic multivariate exponential distribution mean.
windowIntensities	Vector of integrated intensities over all detection windows.
lambda	Rate parameter of the isotropic multivariate exponential distribution.

Value

A matrix of x- and y-coordinates of the generated points. One row corresponds to one point.

Author(s)

Wei Zhang

Examples

```
numPoints <- 10
lowerObsCoords <- matrix(c(0, 0, 1, 0, 0, 1, 1, 1), nrow = 4, byrow = TRUE)
upperObsCoords <- matrix(c(1, 1, 2, 1, 1, 2, 2, 2), nrow = 4, byrow = TRUE)
s <- c(1, 1)
windowIntensities <- c(1:4)
lambda <- 0.1
stratRejectionSampler_exp(numPoints, lowerObsCoords, upperObsCoords, s, windowIntensities, lambda)
```

---

stratRejectionSampler_normal

Stratified rejection sampler for multivariate normal point process

---

Description

Simulate data using a stratified rejection sampler from a point process with an isotropic multivariate normal decay kernel.

Usage

```
stratRejectionSampler_normal(
  numPoints,
  lowerCoords,
  upperCoords,
  s,
  windowIntensities,
  sd
)
```

Arguments

numPoints	Number of spatial points to generate.
lowerCoords, upperCoords	Matrices of lower and upper x- and y-coordinates of a set of detection windows. One row for each window.
s	Vector of x- and y-coordinates of of the isotropic multivariate normal distribution mean.
windowIntensities	Vector of integrated intensities over all detection windows.
sd	Standard deviation of the isotropic multivariate normal distribution.

Value

A matrix of x- and y-coordinates of the generated points. One row corresponds to one point.

Author(s)

Joseph D. Chipperfield and Wei Zhang

Examples

```
numPoints <- 10
lowerObsCoords <- matrix(c(0, 0, 1, 0, 0, 1, 1, 1), nrow = 4, byrow = TRUE)
upperObsCoords <- matrix(c(1, 1, 2, 1, 1, 2, 2, 2), nrow = 4, byrow = TRUE)
s <- c(1, 1)
windowIntensities <- c(1:4)
sd <- 0.1
set.seed(0)
stratRejectionSampler_normal(numPoints, lowerObsCoords, upperObsCoords, s, windowIntensities, sd)
```

Index