Package 'nimbleSCR'

August 20, 2020

Type Package

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

Version 0.1.0

Maintainer Daniel Turek <danielturek@gmail.com>

Date 2020-08-12

Description Provides utility functions, distributions, and methods for improving Markov chain Monte Carlo (MCMC) sampling efficiency for ecological Spatial Capture-Recapture (SCR) and Open Population Spatial Capture-Recapture (OPSCR) models. The methods provided implement the techniques presented in ``Estimation of Large-Scale Spatial Capture-Recapture Models" (Turek, et al (2020) <doi:10.1101/2020.05.07.081182>).

License GPL-3

Depends R (>= 3.4.0), nimble

Encoding UTF-8

LazyData true

RoxygenNote 7.1.1

NeedsCompilation no

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

Repository CRAN

Date/Publication 2020-08-20 09:20:02 UTC

R topics documented:

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

Description

The `dbinom_sparseLocalSCR` distribution is a NIMBLE custom distribution which can be used to model the binomial observations (x) of a single individual over a set of detectors defined by their coordinates (trapCoords). The distribution assumes that the detection probability at any detector follows a half-normal function of the distance between the individual's activity center (s) and the detector location.

Usage

```
dbinom_sparseLocalSCR(
  x,
  detNums,
  detIndices,
  size,
  p0,
  sigma,
  s,
  trapCoords,
  localTrapsIndices,
  localTrapsNum,
  resizeFactor = 1,
  habitatGrid,
  indicator = 1,
  log = 0
)

rbinom_sparseLocalSCR(
  n = 1,
  detNums,
  detIndices,
  size,
  p0,
  sigma,
  s,
  trapCoords,
  localTrapsIndices,
  localTrapsNum,
  resizeFactor = 1,
```

```
  habitatGrid,
  indicator = 1
)
```

Arguments

x	Vector of individual detection frequencies, as returned by the getSparseY function (padded with -1's to maintain the square structure of the observation data).
detNums	Number of detections recorded in x, as returned by the getSparseY function.
detIndices	Vector of the detector indices where the detections in x were recorded, as returned by the getSparseY function.
size	Vector of the number of trials (zero or more) for each trap (trapCoords).
p0	Baseline detection probability used in the half-normal detection function.
sigma	Scale parameter of the half-normal detection function.
s	Bivariate individual activity center coordinates.
trapCoords	Matrix of x- and y-coordinates of all traps.
localTrapsIndices	Matrix of indices of local traps around each habitat grid cell, as returned by the getLocalTraps function.
localTrapsNum	Vector of numbers of local traps around all habitat grid cells, as returned by the getLocalTraps function.
resizeFactor	Aggregation factor used in the getLocalTraps function to reduce the number of habitat grid cells to retrieve local traps for.
habitatGrid	Matrix of habitat grid cells indices.
indicator	Logical argument, specifying whether the individual is available for detection.
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 dbinom_sparseLocalSCR distribution incorporates three features to increase computation efficiency:

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

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 : $p = p0 * exp(-d^2/\sigma^2)$.

Author(s)

Cyril Milleret

Examples

```
## define model code
code <- nimbleCode({
    psi ~ dunif(0,1)
    p0 ~ dunif(0,1)
    sigma ~ dunif(0,100)
    N <- sum(z[1:M])
    for(i in 1:M) {
        s[i, 1] ~ dunif(0, 100)
        s[i, 2] ~ dunif(0, 100)
        z[i] ~ dbern(psi)
        y[i,1:maxDetNum] ~ dbinom_sparseLocalSCR(detNums,
                                                 detIndices,
                                                 size,
                                                 p0,
                                                 sigma,
                                                 s[i,1:2],
                                                 trapCoords,
                                                 localTrapsIndices,
                                                 localTrapsNum,
                                                 resizeFactor,
                                                 habitatGrid,
                                                 z[i])
    }
})

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

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

`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.
```

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

Description

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.

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 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

```
## 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.
```

---

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.
```

---

getLocalTraps *Local Trap Identification*

Description

R utility function to identify all 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 (dbinom_sparseLocalSCR). The distance to the activity center and the detection probability are then calculated for these local traps only (i.e. the detection probability is assumed to be 0 for all other traps as they are far enough from the activity center).

Usage

```
getLocalTraps(
  habitatMask,
  trapCoords,
  dmax,
  resizeFactor = 1,
  plot.check = TRUE
)
```

Arguments

habitatMask	a binary matrix object indicating which cells are considered as suitable habitat.
trapCoords	A matrix giving the x- and y-coordinate of each trap.
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 traps for. Defaults to 1; no aggregation.
plot.check	A visualization option (if TRUE); displays which traps are considered "local traps" for a randomly chosen habitat cell.

Details

The getLocalTraps function is used in advance of model building.

Value

This function returns a list of objects:

- localTrapIndices: 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 traps associated with this grid cell.
- habitatGrid: a matrix of habitat grid cells ID corresponding to the row indices in localTrapIndices.
- numLocalTraps: a vector of the number of local traps for each habitat grid cell in habitatGrid.
- numLocalTrapsMax: the maximum number of local traps 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)
trapCoords <- 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)

localTraps.list <- getLocalTraps(habitatMask, trapCoords, resizeFactor = 1, dmax = 7)
```

---

getSparseY *Sparse Matrix Preparation*

Description

R utility function to turn a two or three-dimensional detection array into a sparse matrix representation. Used in the implementation of the dbinom_sparseLocalSCR function.

Usage

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

Arguments

x	A two- or three-dimensional observation data array.
noDetections	The value indicating no detection. Defaults to -1.

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 vector of the number of traps at which each individual was detected.
- *maxDetNums* the maximum number of traps at which any individual was detected (i.e, the maximum of detNums).
- *detIndices* a matrix of dimensions n.individuals * maxDetNums which contains the IDs of the traps. where each individuals was detected.
- *y* a matrix of dimensions n.individuals * maxDetNums which contains the number of observations of each individual at the traps it was detected at.

Author(s)

Cyril Milleret

Examples

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

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

---

localTrapCalculations *Local Trap Calculations*

Description

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

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

```
## 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.
```

Index