Package ‘nimbleEcology’

February 21, 2020

Type Package

Title Distributions for Ecological Models in 'nimble'

Version 0.2.0

Maintainer Benjamin R. Goldstein <ben.goldstein@berkeley.edu>

Date 2020-02-20

Description


License GPL-3

Copyright Copyright (c) 2019, Perry de Valpine, Ben Goldstein, Daniel Turek, Lauren Ponisio

Depends R (>= 3.4.0), nimble

Encoding UTF-8

LazyData true


RoxygenNote 6.1.1

Suggests rmarkdown, testthat (>= 2.1.0)

NeedsCompilation no

Author Benjamin R. Goldstein [aut, cre],
Daniel Turek [aut],
Lauren Ponisio [aut],
Perry de Valpine [aut]

Repository CRAN

Date/Publication 2020-02-21 17:30:02 UTC
**R topics documented:**

- dCJS ................................................................. 2
- dDHMM .............................................................. 5
- dDynOcc ............................................................ 8
- dHMM ................................................................. 12
- dNmixture .......................................................... 14
- dOcc ................................................................. 17

---

**Description**

- **dCJS_** and **rCJS_** provide Cormack-Jolly-Seber capture-recapture distributions that can be used directly from R or in **nimble** models.

**Usage**

- \texttt{dCJS\_ss(x, probSurvive, probCapture, len = 0, log = 0)}
- \texttt{dCJS\_sv(x, probSurvive, probCapture, len = 0, log = 0)}
- \texttt{dCJS\_vs(x, probSurvive, probCapture, len = 0, log = 0)}
- \texttt{dCJS\_vv(x, probSurvive, probCapture, len = 0, log = 0)}
- \texttt{rCJS\_ss(n, probSurvive, probCapture, len = 0)}
- \texttt{rCJS\_sv(n, probSurvive, probCapture, len = 0)}
- \texttt{rCJS\_vs(n, probSurvive, probCapture, len = 0)}
- \texttt{rCJS\_vv(n, probSurvive, probCapture, len = 0)}

**Arguments**

- **x**
  - capture-history vector of 0s (not captured) and 1s (captured). Include the initial capture, so \texttt{x[1]} should equal 1.
- **probSurvive**
  - survival probability, either a time-independent scalar (for \texttt{dCJS\_s*}) or a time-dependent vector (for \texttt{dCJS\_v*}) with length \texttt{len - 1}.
- **probCapture**
  - capture probability, either a time-independent scalar (for \texttt{dCJS\_*s}) or a time-dependent vector (for \texttt{dCJS\_*v}) with length \texttt{len}. If a vector, first element is ignored, as the total probability is conditioned on the capture at \texttt{t = 1}.
- **len**
  - length of capture history. Should equal \texttt{length(x)}
**Details**

These nimbleFunctions provide distributions that can be used directly in R or in nimble hierarchical models (via `nimbleCode` and `nimbleModel`).

The letters following the 'dCJS_' indicate whether survival and/or capture probabilities, in that order, are scalar (s, meaning the probability applies to every x[t]) or vector (v, meaning the probability is a vector aligned with x). When `probCapture` and/or `probSurvive` is a vector, they must be the same length as x.

It is important to use the time indexing correctly for survival. `probSurvive[t]` is the survival probability from time t to time t + 1. When a vector, `probSurvive` may have length greater than `length(x) - 1`, but all values beyond that index are ignored.

Time indexing for detection is more obvious: `probDetect[t]` is the detection probability at time t.

When called from R, the `len` argument to `dCJS_*` is not necessary. It will default to the length of x. When used in nimble model code (via `nimbleCode`), `len` must be provided (even though it may seem redundant).

For more explanation, see package vignette (or vignette("Introduction_to_nimbleEcology")).

Compared to writing nimble models with a discrete latent state for true alive/dead status at each time and a separate scalar datum for each observation, use of these distributions allows one to directly sum (marginalize) over the discrete latent states and calculate the probability of the detection history for one individual jointly.

These are nimbleFunctions written in the format of user-defined distributions for NIMBLE's extension of the BUGS model language. More information can be found in the NIMBLE User Manual at [https://r-nimble.org](https://r-nimble.org).

When using these distributions in a nimble model, the left-hand side will be used as x, and the user should not provide the `log` argument.

For example, in nimble model code,

```r
captures[i,1:T] ~ dCSJ_ss(survive,capture,T)
```

declares a vector node, `captures[i,1:T]`, (detection history for individual i, for example) that follows a CJS distribution with scalar survival probability `survive` and scalar capture probability `capture` (assuming `survive` and `capture` are defined elsewhere in the model).

This will invoke (something like) the following call to `dCJS_ss` when nimble uses the model such as for MCMC:

```r
dCSJ_ss(captures[i,1:T],survive,capture,len = T,log = TRUE)
```

If an algorithm using a nimble model with this declaration needs to generate a random draw for `captures[i,1:T]`, it will make a similar invocation of `rCJS_ss`, with `n = 1`.

If both survival and capture probabilities are time-dependent, use

```r
captures[i,1:T] ~ dCSJ_vv(survive[1:(T-1)],capture[1:T],T)
```

and so on for each combination of time-dependent and time-independent parameters.
Value

For dCJS_**: the probability (or likelihood) or log probability of observation vector \( x \).

For rCJS_**: a simulated capture history, \( x \).

Author(s)

Ben Goldstein, Perry de Valpine, and Daniel Turek

References


See Also

For multi-state or multi-event capture-recapture models, see dHMM or dDHMM.

Examples

```r
# Set up constants and initial values for defining the model
dat <- c(1,1,0,0,0) # A vector of observations
probSurvive <- c(0.6, 0.3, 0.3, 0.1)
probCapture <- 0.4

# Define code for a nimbleModel
nc <- nimbleCode(
  x[1:4] ~ dCJS_vs(probSurvive, probCapture, len = 4)
  probSurvive ~ dunif(0,1)
  for (i in 1:4) probSurvive[i] ~ dunif(0, 1)
)

# Build the model, providing data and initial values
CJS_model <- nimbleModel(nc, data = list(x = dat),
                         inits = list(probSurvive = probSurvive,
                                      probCapture = probCapture))

# Calculate log probability of data from the model
CJS_model$calculate()
# Use the model for a variety of other purposes...
```
"dDHMM"  Dynamic Hidden Markov Model distribution for use in nimble models

Description

dDHMM and dDHMMo provide Dynamic hidden Markov model distributions for nimble models.

Usage

dDHMM(x, init, probObs, probTrans, len, log = 0)
dDHMMo(x, init, probObs, probTrans, len, log = 0)
rDHMM(n, init, probObs, probTrans, len)
rDHMMo(n, init, probObs, probTrans, len)

Arguments

x  vector of observations, each one a positive integer corresponding to an observation state (one value of which could correspond to "not observed", and another value of which can correspond to "dead" or "removed from system").
init  vector of initial state probabilities. Must sum to 1
probObs  time-independent matrix (dDHMM and rHMM) or time-dependent 3D array (dDHMMo and rHMMo) of observation probabilities. First two dimensions of probObs are of size x (number of possible system states) x (number of possible observation classes). dDHMMo and rHMMo expects an additional third dimension of size (number of observation times)
probTrans  time-dependent array of system state transition probabilities. Dimension of probTrans is (number of possible system states) x (number of possible system states) x (number of observation times)
len  length of observations (needed for rHMM)
log  TRUE or 1 to return log probability. FALSE or 0 to return probability
n  number of random draws, each returning a vector of length len. Currently only n = 1 is supported, but the argument exists for standardization of "r" functions

Details

These nimbleFunctions provide distributions that can be used directly in R or in nimble hierarchical models (via nimbleCode and nimbleModel).

The probability (or likelihood) of observation x[t,o] depends on the previous true latent state, the time-dependent probability of transitioning to a new state probTrans, and the probability of observation states given the true latent state probObs.
The distribution has two forms, `dDHMM` and `dDHMMo`. `dDHMM` takes a time-independent observation probability matrix with dimension S x O, while `dDHMMo` expects a three-dimensional array of time-dependent observation probabilities with dimension S x O x T, where O is the number of possible occupancy states, S is the number of true latent states, and T is the number of time intervals.

`probTrans` has dimension S x S x (T - 1). `probTrans[i, j, t]` is the probability that an individual in state i at time t takes on state j at time t+1. The length of the third dimension may be greater than (T - 1) but all values indexed greater than T - 1 will be ignored.

`initStates` has length S. `initStates[i]` is the probability of being in state i at the first observation time.

For more explanation, see package vignette (or vignette("Introduction_to_nimbleEcology")). Compared to writing `nimble` models with a discrete true latent state and a separate scalar datum for each observation, use of these distributions allows one to directly sum (marginalize) over the discrete latent state and calculate the probability of all observations from one site jointly.

These are `nimbleFunctions` written in the format of user-defined distributions for NIMBLE’s extension of the BUGS model language. More information can be found in the NIMBLE User Manual at https://r-nimble.org.

When using these distributions in a `nimble` model, the left-hand side will be used as x, and the user should not provide the log argument.

For example, in a NIMBLE model,

```r
observedStates[1:T] ~ dDHMM(initStates[1:S],observationProbs[1:S,1:O],transitionProbs[1:S,1:S,1:(T-1)],T)
```

declares that the `observedStates[1:T]` vector follows a dynamic hidden Markov model distribution with parameters as indicated, assuming all the parameters have been declared elsewhere in the model. In this case, S is the number of system states, O is the number of observation classes, and T is the number of observation occasions. This will invoke (something like) the following call to `dDHMM` when `nimble` uses the model such as for MCMC:

```r
rDHMM(observedStates[1:T],initStates[1:S],observationProbs[1:S,1:O],transitionProbs[1:S,1:S,1:(T-1)],T)
```

If an algorithm using a `nimble` model with this declaration needs to generate a random draw for `observedStates[1:T]`, it will make a similar invocation of `rDHMM`, with n = 1.

If the observation probabilities are time-dependent, one would use:

```r
```

Value

For `dDHMM` and `dDHMMo`: the probability (or likelihood) or log probability of observation vector x.

For `rDHMM` and `rDHMMo`: a simulated detection history, x.

Author(s)

Perry de Valpine, Daniel Turek, and Ben Goldstein

References

dDHMM

See Also

For hidden Markov models with time-independent transitions, see dHMM and dHMMo. For simple capture-recapture, see dCJS.

Examples

# Set up constants and initial values for defining the model
dat <- c(1,2,1,1) # A vector of observations
init <- c(0.4, 0.2, 0.4) # A vector of initial state probabilities
probObs <- t(array(# A matrix of observation probabilities
  c(1, 0,
    0, 1,
    0.8, 0.2), c(2, 3)))
probTrans <- array(rep(0.5, 27), # A matrix of time-indexed transition probabilities
  c(3,3,3))

# Define code for a nimbleModel
nc <- nimbleCode(
  x[1:4] ~ dDHMM(init[1:3], probObs = probObs[1:3, 1:2],
    probTrans = probTrans[1:3, 1:3, 1:4], len = 4)
  for (i in 1:3) {
    init[i] ~ dunif(0,1)
    for (j in 1:3) {
      for (t in 1:4) {
        probTrans[i,j,t] ~ dunif(0,1)
      }
    }
    probObs[i, 1] ~ dunif(0,1)
    probObs[i, 2] <- 1 - probObs[1,i]
  }
)

# Build the model, providing data and initial values
DHMM_model <- nimbleModel(nc,
  data = list(x = dat),
  inits = list(init = init,
    probObs = probObs,
    probTrans = probTrans))

# Calculate log probability of x from the model
DHMM_model$calculate()
# Use the model for a variety of other purposes...
dDynOcc

*Dynamic occupancy distribution for use in nimble models*

dDynOcc\_\*\* and rDynOcc\_\*\* provide dynamic occupancy model distributions that can be used directly from R or in nimble models.

**Description**

Dynamic occupancy distribution for use in nimble models dDynOcc\_\*\* and rDynOcc\_\*\* provide dynamic occupancy model distributions that can be used directly from R or in nimble models.

**Usage**

dDynOcc\_vvm(x, init, probPersist, probColonize, p, start, end, log = 0)
dDynOcc\_vsm(x, init, probPersist, probColonize, p, start, end, log = 0)
dDynOcc\_svm(x, init, probPersist, probColonize, p, start, end, log = 0)
dDynOcc\_ssm(x, init, probPersist, probColonize, p, start, end, log = 0)
rDynOcc\_vvm(n, init, probPersist, probColonize, p, start, end)
rDynOcc\_vsm(n, init, probPersist, probColonize, p, start, end)
rDynOcc\_svm(n, init, probPersist, probColonize, p, start, end)
rDynOcc\_ssm(n, init, probPersist, probColonize, p, start, end)
dDynOcc\_vvv(x, init, probPersist, probColonize, p, start, end, log = 0)
dDynOcc\_vsv(x, init, probPersist, probColonize, p, start, end, log = 0)
dDynOcc\_svv(x, init, probPersist, probColonize, p, start, end, log = 0)
dDynOcc\_ssv(x, init, probPersist, probColonize, p, start, end, log = 0)
rDynOcc\_vvv(n, init, probPersist, probColonize, p, start, end)
rDynOcc\_vsv(n, init, probPersist, probColonize, p, start, end)
rDynOcc\_svv(n, init, probPersist, probColonize, p, start, end)
rDynOcc\_ssv(n, init, probPersist, probColonize, p, start, end)
dDynOcc\_vvs(x, init, probPersist, probColonize, p, start, end, log = 0)
dDynOcc\_vss(x, init, probPersist, probColonize, p, start, end, log = 0)


dDynOcc_svs(x, init, probPersist, probColonize, p, start, end, log = 0)
dDynOcc_sss(x, init, probPersist, probColonize, p, start, end, log = 0)
rDynOcc_vvs(n, init, probPersist, probColonize, p, start, end)
rDynOcc_vss(n, init, probPersist, probColonize, p, start, end)
rDynOcc_svs(n, init, probPersist, probColonize, p, start, end)
rDynOcc_sss(n, init, probPersist, probColonize, p, start, end)

Arguments

x detection/non-detection matrix of 0s (not detected) and 1s (detected). Rows represent primary sampling occasions (e.g. different seasons). Columns are secondary sampling locations (e.g. replicate visits within a season) that may be different for each row

init probability of occupancy in the first sampling period

probPersist persistence probability–probability an occupied cell remains occupied. 1-extinction probability. Scalar for dDynOcc_s**, vector for dDynOcc_v**. If vector, should have length dim(x)[1] - 1 since no transition occurs after the last observation

probColonize colonization probability. Probability that an unoccupied cell becomes occupied. Scalar for dDynOcc_**s, vector for dDynOcc_**v. If vector, should have length dim(x)[1] - 1 since no transition occurs after the last observation

p Detection probabilities. Scalar for dDynOcc_**s, vector for dDynOcc_**v, matrix for dDynOcc_**m. If a matrix, dimensions should match x

start indicates the column number of the first observation in each row of x. A vector of length dim(x)[1]. This allows for different time periods to have different numbers of sampling occasions

end indicates the column number of the last observation in each row of x. A vector of length dim(x)[1]. This allows for different time periods to have different numbers of sampling occasions

log TRUE (return log probability) or FALSE (return probability)

n number of random draws, each returning a matrix of dimension c(min(start), max(end)). Currently only n = 1 is supported, but the argument exists for standardization of “r” functions

Details

These nimbleFunctions provide distributions that can be used directly in R or in nimble hierarchical models (via nimbleCode and nimbleModel).

The probability (or likelihood) of observation x[t,o] depends on the occupancy status of the site at time t-1, the transition probability of persistence (probPersist or probPersist[t]), colonization (probColonize or probColonize[t]), and a detection probability (p, p[t], or p[t,o]).
The first two letters following the 'dDynOcc_' indicate whether the probabilities of persistence and colonization are a constant scalar (s) or time-indexed vector (v). For example, dDynOcc_svm takes scalar persistence probability probPersist with a vector of colonization probabilities probColonize[1:(T-1)].

When vectors, probColonize and probPersist may be of any length greater than length(x) - 1. Only the first length(x) - 1 indices are used, each corresponding to the transition from time t to t+1 (e.g. probColonize[2] describes the transition probability from t = 2 to t = 3). All extra values are ignored. This is to make it easier to use one distribution for many sites, some requiring probabilities of length 1.

The third letter in the suffix indicates whether the detection probability is a constant (scalar), time-dependent (vector), or both time-dependent and dependent on observation occasion (matrix). For example, dDyn0cc_svm takes a matrix of detection probabilities p[1:T,1:O].

The arguments start and end allow different time periods to contain different numbers of sampling events. Suppose you have observations for samples in three seasons; in the first two seasons, there are four observations, but in the third, there are only three. The start and end could be provided as start = c(1,1,1) and end = c(4,4,3). In this case, the value of x[4,4] would be ignored.

For more explanation, see package vignette (or vignette("Introduction_to_nimbleEcology")). Compared to writing nimble models with a discrete latent state for true occupancy status and a separate scalar datum for each observation, use of these distributions allows one to directly sum (marginalize) over the discrete latent state and calculate the probability of all observations from one site jointly.

These are nimbleFunctions written in the format of user-defined distributions for NIMBLE's extension of the BUGS model language. More information can be found in the NIMBLE User Manual at https://r-nimble.org.

When using these distributions in a nimble model, the left-hand side will be used as x, and the user should not provide the log argument.

For example, in nimble model code,

detections[1:T,1:O] ~ dDynOcc_ssm(init, probPersist = persistence_prob, probColonize = colonization_prob, p = p[1:T,1:O], start = start[1:T], end = end[1:T])

declares that the detections[1:T] vector follows a dynamic occupancy model distribution with parameters as indicated, assuming all the parameters have been declared elsewhere in the model. This will invoke (something like) the following call to dDynOcc_ssm when nimble uses the model such as for MCMC:

dDynOcc_ssm(detections[1:T,1:O], init, probPersist = persistence_prob, probColonize = colonization_prob, p = p[1:T,1:O], start = start[1:T], end = end[1:T], log = TRUE)

If an algorithm using a nimble model with this declaration needs to generate a random draw for detections[1:T,1:O], it will make a similar invocation of rDynOcc_svm, with n = 1.

If the colonization probabilities are time-dependent, one would use:

detections[1:T] ~ dDynOcc_svm(nrep, init = init_prob, probPersist = persistence_prob, probColonize = colonization_prob[1:(T-1)], p = p[1:S,1:T])

Value

For dDynOcc_***: the probability (or likelihood) or log probability of observation vector x. For rDynOcc_***: a simulated detection history, x.
\textit{dDynOcc}

\textbf{Author(s)}
Ben Goldstein, Perry de Valpine and Lauren Ponisio

\textbf{See Also}
For basic occupancy models, see documentation for \textit{dOcc}.

\textbf{Examples}

\begin{verbatim}
# Set up constants and initial values for defining the model
x <- matrix(c(0,0,NA,0,
              1,1,1,0,
              0,0,0,0,
              0,0,1,0,
              0,0,0,NA), nrow = 4)
start <- c(1,1,2,1)
end <- c(5,5,5,4)
init <- 0.7
probPersist <- 0.5
probColonize <- 0.2
p <- 0.8

# Define code for a nimbleModel
nc <- nimbleCode(
  x[1:2, 1:5] ~ dDynOcc_vvm(nrep[1:2], init,
                           probPersist[1:2], probColonize[1:2], p[1:2,1:5])
  init ~ dunif(0,1)
  for (i in 1:2) {
    probPersist[i] ~ dunif(0,1)
    probColonize[i] ~ dunif(0,1)
  }
  for (i in 1:2) {
    for (j in 1:5) {
      p[i,j] ~ dunif(0,1)
    }
  }
)

# Build the model, providing data and initial values
DynOcc_model <- nimbleModel(nc, data = list(x = dat, nrep = nrep),
                             inits = list(p = p, probPersist = probPersist,
                                          init = init, probColonize = probColonize))

# Calculate log probability of data from the model
DynOcc_model$calculate()
# Use the model for a variety of other purposes...
\end{verbatim}
**dHMM**

*Hidden Markov Model distribution for use in nimble models*

**Description**

`dHMM` and `dHMMo` provide hidden Markov model distributions that can be used directly from R or in `nimble` models.

**Usage**

- `dHMM(x, init, probObs, probTrans, len = 0, log = 0)`
- `dHMMo(x, init, probObs, probTrans, len = 0, log = 0)`
- `rHMM(n, init, probObs, probTrans, len = 0)`
- `rHMMo(n, init, probObs, probTrans, len = 0)`

**Arguments**

- **x**
  - vector of observations, each one a positive integer corresponding to an observation state (one value of which could correspond to "not observed", and another value of which can correspond to "dead" or "removed from system").
- **init**
  - vector of initial state probabilities. Must sum to 1.
- **probObs**
  - time-independent matrix (`dHMM` and `rHMM`) or time-dependent array (`dHMMo` and `rHMMo`) of observation probabilities. First two dimensions of `probObs` are of size x (number of possible system states) x (number of possible observation classes). `dHMMo` and `rHMMo` expects an additional third dimension of size (number of observation times).
- **probTrans**
  - time-independent matrix of state transition probabilities.
- **len**
  - length of x (see below).
- **log**
  - TRUE or 1 to return log probability. FALSE or 0 to return probability.
- **n**
  - number of random draws, each returning a vector of length len. Currently only n = 1 is supported, but the argument exists for standardization of "r" functions.

**Details**

These nimbleFunctions provide distributions that can be used directly in R or in nimble hierarchical models (via `nimbleCode` and `nimbleModel`).

The distribution has two forms, `dHMM` and `dHMMo`. Define S as the number of latent state categories (maximum possible value for elements of x), O as the number of possible observation state categories, and T as the number of observation times (length of x). In `dHMM`, `probObs` is a time-independent observation probability matrix with dimension S x O. In `dHMMo`, `probObs` is a three-dimensional array of time-dependent observation probabilities with dimension S x O x T. The first
index of \texttt{probObs} indexes the true latent state. The second index of \texttt{probObs} indexes the observed state. For example, in the time-dependent case, \texttt{probObs[i,j,t]} is the probability at time \(t\) that an individual in state \(i\) is observed in state \(j\).

\texttt{probTrans} has dimension \(S \times S\). \texttt{probTrans[i,j]} is the tune-independent probability that an individual in state \(i\) at time \(t\) transitions to state \(j\) time \(t+1\).

\texttt{initStates} has length \(S\). \texttt{initStates[i]} is the probability of being in state \(i\) at the first observation time.

For more explanation, see \texttt{package vignette} (or \texttt{vignette("Introduction_to_nimbleEcology")}). Compared to writing \texttt{nimble} models with a discrete latent state and a separate scalar datum for each observation time, use of these distributions allows one to directly sum (marginalize) over the discrete latent state and calculate the probability of all observations for one individual (or other HMM unit) jointly.

These are \texttt{nimbleFunctions} written in the format of user-defined distributions for NIMBLE's extension of the BUGS model language. More information can be found in the NIMBLE User Manual at \url{https://r-nimble.org}.

When using these distributions in a \texttt{nimble} model, the left-hand side will be used as \(x\), and the user should not provide the \texttt{log} argument.

For example, in \texttt{nimble} model code,

\begin{verbatim}
observedStates[1,1:T] ~ dHMM(initStates[1:S],observationProbs[1:S,1:O],transitionProbs[1:S,1:S],T)
\end{verbatim}

declares that the \texttt{observedStates[1,1:T]} (observation history for individual \(i\), for example) vector follows a hidden Markov model distribution with parameters as indicated, assuming all the parameters have been declared elsewhere in the model. As above, \(S\) is the number of system state categories, \(O\) is the number of observation state categories, and \(T\) is the number of observation occasions. This will invoke (something like) the following call to \texttt{dHMM} when \texttt{nimble} uses the model such as for MCMC:

\begin{verbatim}
dHMM(observedStates[1:T],initStates[1:S],observationProbs[1:S,1:O],transitionProbs[1:S,1:S],T,log = TRUE)
\end{verbatim}

If an algorithm using a \texttt{nimble} model with this declaration needs to generate a random draw for \texttt{observedStates[1:T]}, it will make a similar invocation of \texttt{rHMM}, with \(n = 1\).

If the observation probabilities are time-dependent, one would use:

\begin{verbatim}
observedStates[1:T] ~ dHMMo(initStates[1:S],observationProbs[1:O,1:S,1:T],transitionProbs[1:S,1:S],T)
\end{verbatim}

\textbf{Value}

For \texttt{dHMM} and \texttt{dHMMo}: the probability (or likelihood) or log probability of observation vector \(x\).

For \texttt{rHMM} and \texttt{rHMMo}: a simulated detection history, \(x\).

\textbf{Author(s)}

Ben Goldstein, Perry de Valpine, and Daniel Turek

\textbf{References}

See Also

For dynamic hidden Markov models with time-dependent transitions, see `dDHMM` and `dDHMMo`. For simple capture-recapture, see `dCJS`.

Examples

```r
# Set up constants and initial values for defining the model
len <- 5 # length of dataset
dat <- c(1,2,1,1,2) # A vector of observations
init <- c(0.4, 0.2, 0.4) # A vector of initial state probabilities
probObs <- t(array(
  c(1, 0,
   0, 1,
   0.2, 0.8), c(2, 3)))
probTrans <- t(array(
  c(0.6, 0.3, 0.1,
   0, 0.7, 0.3,
   0, 0, 1), c(3,3)))

# Define code for a nimbleModel
nc <- nimbleCode(
  x[1:5] ~ dHMM(init[1:3], probObs = probObs[1:2,1:3],
              probTrans = probTrans[1:3, 1:3], len = 5)
  for (i in 1:3) {
    init[i] ~ dunif(0,1)
    for (j in 1:3) {
      probTrans[i,j] ~ dunif(0,1)
    }
    probObs[i, 1] ~ dunif(0,1)
    probObs[i, 2] <- 1 - probObs[1,i]
  }
)

# Build the model
HMM_model <- nimbleModel(nc,
data = list(x = dat),
inits = list(init = init,
probObs = probObs,
probTrans = probTrans))

# Calculate log probability of data from the model
HMM_model$calculate()
# Use the model for a variety of other purposes...
```

dNmix {N-mixture distribution for use in nimble models}
**Description**

\texttt{dNmixture\_s} and \texttt{dNmixture\_v} provide Poisson-Binomial mixture distributions of abundance ("N-mixture") for use in \texttt{nimble} models.

**Usage**

\begin{verbatim}
dNmixture\_v(x, lambda, prob, Nmin = -1, Nmax = -1, len, log = 0)
dNmixture\_s(x, lambda, prob, Nmin = -1, Nmax = -1, len, log = 0)
rNmixture\_v(n, lambda, prob, Nmin = -1, Nmax = -1, len)
rNmixture\_s(n, lambda, prob, Nmin = -1, Nmax = -1, len)
\end{verbatim}

**Arguments**

- \texttt{x} vector of integer counts from a series of sampling occasions.
- \texttt{lambda} expected value of the Poisson distribution of true abundance
- \texttt{prob} detection probability (scalar for \texttt{dNmixture\_s}, vector for \texttt{dNmixture\_v}).
- \texttt{Nmin} minimum abundance to sum over for the mixture probability. Set to -1 to select automatically.
- \texttt{Nmax} maximum abundance to sum over for the mixture probability. Set to -1 to select automatically.
- \texttt{len} The length of the \texttt{x} vector
- \texttt{log} TRUE or 1 to return log probability. FALSE or 0 to return probability.
- \texttt{n} number of random draws, each returning a vector of length \texttt{len}. Currently only \texttt{n = 1} is supported, but the argument exists for standardization of "r" functions.

**Details**

These \texttt{nimbleFunctions} provide distributions that can be used directly in R or in \texttt{nimble} hierarchical models (via \texttt{nimbleCode} and \texttt{nimbleModel}).

An N-mixture model defines a distribution for multiple counts (typically of animals, typically made at a sequence of visits to the same site). The latent number of animals available to be counted, \(N\), follows a Poisson distribution with mean \(\lambda\). Each count, \(x[i]\) for visit \(i\), follows a binomial distribution with size (number of trials) \(N\) and probability of success (being counted) \(\text{prob}[i]\).

The distribution has two forms, \texttt{dNmixture\_s} and \texttt{dNmixture\_v}. With \texttt{dNmixture\_s}, detection probability is a scalar, independent of visit, so \(\text{prob}[i]\) should be replaced with \(\text{prob}\) above. With \texttt{dNmixture\_v}, detection probability is a vector, with one element for each visit, as written above.

For more explanation, see package vignette (or vignette("Introduction\_to\_nimbleEcology")). Compared to writing \texttt{nimble} models with a discrete latent state of abundance \(N\) and a separate scalar datum for each count, use of these distributions allows one to directly sum (marginalize) over the discrete latent state \(N\) and calculate the probability of all observations for a site jointly.

If one knows a reasonable range for summation over possible values of \(N\), the start and end of the range can be provided as \texttt{Nmin} and \texttt{Nmax}. Otherwise one can set both to -1, in which case values for
Nmin and Nmax will be chosen based on the 0.0001 and 0.9999 quantiles of the marginal distributions of each count, using the minimum over counts of the former and the maximum over counts of the latter.

The summation over N uses the efficient method given by Meehan et al. (2017).

These are nimbleFunctions written in the format of user-defined distributions for NIMBLE’s extension of the BUGS model language. More information can be found in the NIMBLE User Manual at https://r-nimble.org.

When using these distributions in a nimble model, the left-hand side will be used as x, and the user should not provide the log argument.

For example, in nimble model code,

\[ \text{observedCounts}[i,1:T] \sim \text{dNmixture}_v(\text{lambda}[i], \text{prob}[i,1:T], \text{Nmin}, \text{Nmax}, T) \]

declares that the observedCounts[i,1:T] (observed counts for site i, for example) vector follows an N-mixture distribution with parameters as indicated, assuming all the parameters have been declared elsewhere in the model. As above, \( \text{lambda}[i] \) is the mean of the abundance distribution at site i, \( \text{prob}[i,1:T] \) is a vector of detection probabilities at site i, and T is the number of observation occasions. This will invoke (something like) the following call to dNmixture_v when nimble uses the model such as for MCMC:

\[ \text{dNmixture}_v(\text{observedCounts}[i,1:T], \text{lambda}[i], \text{prob}[i,1:T], \text{Nmin}, \text{Nmax}, T, \text{log} = \text{TRUE}) \]

If an algorithm using a nimble model with this declaration needs to generate a random draw for observedCounts[1:T], it will make a similar invocation of rNmixture_v, with \( n = 1 \).

If the observation probabilities are visit-independent, one would use:

\[ \text{observedCounts}[1:T] \sim \text{dNmixture}_s(\text{observedCounts}[i,1:T], \text{lambda}[i], \text{prob}[i], \text{Nmin}, \text{Nmax}, T) \]

**Value**

For dNmixture_s and dNmixture_v: the probability (or likelihood) or log probability of observation vector x.

For rNmixture_s and rNmixture_v: a simulated detection history, x.

**Author(s)**

Ben Goldstein, Lauren Ponisio, and Perry de Valpine

**References**


**See Also**

For occupancy models dealing with detection/nondetection data, see dOcc. For dynamic occupancy, see dDynOcc.
Examples

# Set up constants and initial values for defining the model
len <- 5 # length of dataset
dat <- c(1,2,0,1,5) # A vector of observations
lambda <- 10 # mean abundance
prob <- c(0.2, 0.3, 0.2, 0.1, 0.4) # A vector of detection probabilities

# Define code for a nimbleModel
nc <- nimbleCode(
  x[1:5] ~ dNmixture_v(lambda, prob = prob[1:5],
    Nmin = -1, Nmax = -1, len = 5)

  lambda ~ dunif(0, 1000)

  for (i in 1:5) {
    prob[i] ~ dunif(0, 1)
  }
)

# Build the model
nmix <- nimbleModel(nc,
  data = list(x = dat),
  inits = list(lambda = lambda,
    prob = prob))

# Calculate log probability of data from the model
nmix_model$calculate()
# Use the model for a variety of other purposes...

---

**dOcc**  
*Occupancy distribution suitable for use in nimble models*

### Description

dOcc_* and rOcc_* provide occupancy model distributions that can be used directly from R or in nimble models.

### Usage

```
dOcc_s(x, probOcc, probDetect, len = 0, log = 0)
dOcc_v(x, probOcc, probDetect, len = 0, log = 0)
rOcc_s(n, probOcc, probDetect, len = 0)
rOcc_v(n, probOcc, probDetect, len = 0)
```
Arguments

- **x**: detection/non-detection vector of 0s (not detected) and 1s (detected).
- **probOcc**: occupancy probability (scalar).
- **probDetect**: detection probability (scalar for \(d\text{Occ}_s\), vector for \(d\text{Occ}_v\)).
- **len**: length of detection/non-detection vector (see below).
- **log**: TRUE or 1 to return log probability. FALSE or 0 to return probability.
- **n**: number of random draws, each returning a vector of length \(len\). Currently only \(n = 1\) is supported, but the argument exists for standardization of "r" functions.

Details

These nimbleFunctions provide distributions that can be used directly in R or in nimble hierarchical models (via nimbleCode and nimbleModel).

The probability of observation vector \(x\) depends on occupancy probability, \(\text{probOcc}\), and detection probability, \(\text{probDetect}\) or \(\text{probDetect}[t]\).

The letter following the 'dOcc_' indicates whether detection probability is scalar (s, meaning \(\text{probDetect}\) is detection probability for every \(x[t]\)) or vector (v, meaning \(\text{probDetect}[t]\) is detection probability for \(x[t]\)).

When used directly from R, the \(len\) argument to \(d\text{Occ}_*\) is not necessary. It will default to the length of \(x\). When used in nimble model code (via nimbleCode), \(len\) must be provided (even though it may seem redundant).

For more explanation, see package vignette (or vignette("Introduction_to_nimbleEcology")).

Compared to writing nimble models with a discrete latent state for true occupancy status and a separate scalar datum for each observation, use of these distributions allows one to directly sum (marginalize) over the discrete latent state and calculate the probability of all observations from one site jointly.

These are nimbleFunctions written in the format of user-defined distributions for NIMBLE’s extension of the BUGS model language. More information can be found in the NIMBLE User Manual at https://r-nimble.org.

When using these distributions in a nimble model, the left-hand side will be used as \(x\), and the user should not provide the \(log\) argument.

For example, in nimble model code,

\[
detections[i,1:T] \sim d\text{Occ}_s(\text{occupancyProbability}, \text{detectionProbability}, T)
\]

declares that \(detections[i,1:T]\) (detection history at site \(i\), for example) follows an occupancy distribution with parameters as indicated, assuming all the parameters have been declared elsewhere in the model. This will invoke (something like) the following call to \(d\text{Occ}_s\) when nimble uses the model such as for MCMC:

\[
d\text{Occ}_s(detections[i,1:T], \text{occupancyProbability}, \text{detectionProbability}, len = T, \text{log} = \text{TRUE})
\]

If an algorithm using a nimble model with this declaration needs to generate a random draw for \(detections[i,1:T]\), it will make a similar invocation of \(r\text{Occ}_s\), with \(n = 1\).

If the detection probabilities are time-dependent, use:

\[
detections[i,1:T] \sim d\text{Occ}_v(\text{occupancyProbability}, \text{detectionProbability}[1:T], \text{len} = T)
\]
Value

For `dOcc_*`: the probability (or likelihood) or log probability of observation vector `x`.
For `rOcc_*`: a simulated detection history, `x`.

Author(s)

Ben Goldstein, Perry de Valpine, and Lauren Ponisio

See Also

For dynamic occupancy models, see documentation for `dDynOcc`.

Examples

```r
# Set up constants and initial values for defining the model
dat <- c(1,1,0,0) # A vector of observations
probOcc <- 0.6
probDetect <- 0.4

# Define code for a nimbleModel
nc <- nimbleCode(
  x[1:4] ~ dOcc_s(probOcc, probDetect, len = 4)
  probOcc ~ dunif(0,1)
  probDetect ~ dunif(0,1)
)

# Build the model, providing data and initial values
Occ_model <- nimbleModel(nc, data = list(x = dat),
  inits = list(probOcc = probOcc,
  probDetect = probDetect))

# Calculate log probability of data from the model
Occ_model$calculate()
# Use the model for a variety of other purposes...
```
Index

dCJS, 2, 7, 14
dCJS_ss (dCJS), 2
dCJS_sv (dCJS), 2
dCJS_vs (dCJS), 2
dCJS_vv (dCJS), 2
dDHMM, 4, 5, 14
dDHMMo, 14
dDHMMo (dDHMM), 5
dDynOcc, 8, 16, 19
dDynOcc_ssm (dDynOcc), 8
dDynOcc_sss (dDynOcc), 8
dDynOcc_ssv (dDynOcc), 8
dDynOcc_svm (dDynOcc), 8
dDynOcc_ssvs (dDynOcc), 8
dDynOcc_vsm (dDynOcc), 8
dDynOcc_vss (dDynOcc), 8
dDynOcc_vsv (dDynOcc), 8
dDynOcc_vvm (dDynOcc), 8
dDynOcc_vvs (dDynOcc), 8
dDynOcc_vvv (dDynOcc), 8
dHMM, 4, 7, 12
dHMMo, 7
dHMMo (dHMM), 12
dNmixture, 14
dNmixture_s (dNmixture), 14
dNmixture_v (dNmixture), 14
dOcc, 11, 16, 17
dOcc_s (dOcc), 17
dOcc_v (dOcc), 17

nimbleCode, 3, 5, 9, 12, 15, 18
nimbleModel, 3, 5, 9, 12, 15, 18

rCJS_ss (dCJS), 2
rCJS_sv (dCJS), 2
rCJS_vs (dCJS), 2
rCJS_vv (dCJS), 2
rDHMM (dDHMM), 5
rDHMMo (dDHMM), 5

rDynOcc_ssm (dDynOcc), 8
rDynOcc_sss (dDynOcc), 8
rDynOcc_ssv (dDynOcc), 8
rDynOcc_svm (dDynOcc), 8
rDynOcc_svs (dDynOcc), 8
rDynOcc_svsv (dDynOcc), 8
rDynOcc_vsm (dDynOcc), 8
rDynOcc_vss (dDynOcc), 8
rDynOcc_vsv (dDynOcc), 8
rDynOcc_vvm (dDynOcc), 8
rDynOcc_vvs (dDynOcc), 8
rDynOcc_vvv (dDynOcc), 8
rHMM (dHMM), 12
rHMMo (dHMM), 12
rNmixture_s (dNmixture), 14
rNmixture_v (dNmixture), 14
rOcc_s (dOcc), 17
rOcc_v (dOcc), 17