Package ‘nimbleEcology’

October 9, 2019

**Type** Package

**Title** Distributions for Ecological Models in 'nimble'

**Version** 0.1.0

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

**Date** 2019-09-24

**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

**URL** https://github.com/nimble-dev/nimbleEcology

**Collate** dDynOcc.R dCJS.R dDHMM.R dHMM.R dOcc.R zzz.R

**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** 2019-10-09 08:40:07 UTC
**R topics documented:**

- `dCJS` .......................................................... 2
- `dDHMM` .......................................................... 4
- `dDynOcc` ....................................................... 7
- `dHMM` ........................................................ 11
- `dOcc` .......................................................... 14

**Index**

- `dCJS` Cormack-Jolly-Seber distribution for use in nimble models

**Description**

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

**Usage**

- `dCJS_ss(x, probSurvive, probCapture, len = 0, log = 0)`
- `dCJS_sv(x, probSurvive, probCapture, len = 0, log = 0)`
- `dCJS_vs(x, probSurvive, probCapture, len = 0, log = 0)`
- `dCJS_vv(x, probSurvive, probCapture, len = 0, log = 0)`
- `rCJS_ss(n, probSurvive, probCapture, len = 0)`
- `rCJS_sv(n, probSurvive, probCapture, len = 0)`
- `rCJS_vs(n, probSurvive, probCapture, len = 0)`
- `rCJS_vv(n, probSurvive, probCapture, len = 0)`

**Arguments**

- `x` capture-history vector of 0s (not captured) and 1s (captured). Do not include the initial capture, which is assumed to have occurred prior to `x[1]`.
- `probSurvive` survival probability, either a time-independent scalar (for `dCJS_s*`) or a time-dependent vector (for `dCJS_v*`).
- `probCapture` capture probability, either a time-independent scalar (for `dCJS_*s`) or a time-dependent vector (for `dCJS_*v`).
- `len` length of capture history. Should equal `length(x)`
- `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.
**dCJS**

**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 \(t-1\) to time \(t\). 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.

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,

```nimble
captures[i,1:T] ~ dCJS_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:

```nimble
dCJS_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

```nimble
captures[i,1:T] ~ dCJS_vv(survive[1:T],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)  # A vector of observations
probSurvive <- 0.6
probCapture <- 0.4

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

# Build the model, providing data and initial values
CJS_model <- nimbleModel(code, 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 rDHMM)
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.

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,

\[
\text{observedStates}[1:T] \sim dDHMM(\text{initStates}[1:S], \text{observationProbs}[1:S,1:O], \text{transitionProbs}[1:S,1:S,1:(T-1)], T)
\]

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 \text{dDHMM} when \text{nimble} uses the model such as for MCMC:

\[
\text{rDHMM}\left(\text{observedStates}[1:T], \text{initStates}[1:S], \text{observationProbs}[1:S,1:O], \text{transitionProbs}[1:S,1:S,1:(T-1)], T, \log = \text{TRUE}\right)
\]

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 \text{rDHMM}, with \text{n = 1}.

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

\[
\text{observedStates}[1:T] \sim dDHMMo(\text{initStates}[1:S], \text{observationProbs}[1:S,1:O,1:(T-1)], \text{transitionProbs}[1:S,1:S,1:(T-1)], T)
\]

Value

For \text{dDHMM} and \text{dDHMMo}: the probability (or likelihood) or log probability of observation vector x.

For \text{rDHMM} and \text{rDHMMo}: a simulated detection history, x.

Author(s)

Perry de Valpine, Daniel Turek, and Ben Goldstein

References


See Also

For hidden Markov models with time-independent transitions, see \text{dHMM} and \text{dHMMo}. For simple capture-recapture, see \text{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**

```r
# Dynamic occupancy distributions

rDynOcc_vvm(n, init, probPersist, probColonize, p, start, end)
```

```r
# Wrapper for C++ implementation

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)
```

```r
# Dynamic occupancy distributions

rDynOcc_vvv(n, init, probPersist, probColonize, p, start, end)
```

```r
# Wrapper for C++ implementation

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)
```

```r
# Dynamic occupancy distributions

rDynOcc_vvs(n, init, probPersist, probColonize, p, start, end)
```

```r
# Wrapper for C++ implementation

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]`.

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, `dDynOcc_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,

```nimble
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:

```nimble
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:

```nimble
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`.

**Author(s)**

Ben Goldstein, Perry de Valpine and Lauren Ponisio

**See Also**

For basic occupancy models, see documentation for `dOcc`. 
Examples

# 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...
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 probObs indexes the true latent state. The second index of probObs indexes the observed state. For example, in the time-dependent case, probObs[i, j, t] is the probability at time t that an individual in state i is observed in state j.

probTrans has dimension S x S. probTrans[i, j] is the time-independent probability that an individual in state i at time t transitions to state j time t+1.
initStates has length $S$. $\text{initStates}[i]$ 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 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 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,

$$\text{observedStates}[i,1:T] \sim d\text{HMM}(\text{initStates}[1:S],\text{observationProbs}[1:S,1:O],\text{transitionProbs}[1:S,1:S],T)$$

declares that the $\text{observedStates}[i,1:T]$ (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 $d\text{HMM}$ when nimble uses the model such as for MCMC:

$$d\text{HMM}(\text{observedStates}[1:T],\text{initStates}[1:S],\text{observationProbs}[1:S,1:O],\text{transitionProbs}[1:S,1:S],T,\text{log} = \text{TRUE})$$

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

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

$$\text{observedStates}[1:T] \sim d\text{HMMo}(\text{initStates}[1:S],\text{observationProbs}[1:O,1:S,1:T],\text{transitionProbs}[1:S,1:S],T)$$

Value

For $d\text{HMM}$ and $d\text{HMMo}$: the probability (or likelihood) or log probability of observation vector $x$.

For $r\text{HMM}$ and $r\text{HMMo}$: a simulated detection history, $x$.

Author(s)

Ben Goldstein, Perry de Valpine, and Daniel Turek

References


See Also

For dynamic hidden Markov models with time-dependent transitions, see $d\text{DHMM}$ and $d\text{DHMMo}$. For simple capture-recapture, see $d\text{CJS}$. 
Examples

# 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( # A matrix of observation probabilities
  c(1, 0,
  0, 1,
  0.2, 0.8), c(2, 3)))
probTrans <- t(array( # A matrix of transition probabilities
  c(0.6, 0.3, 0.1,
  0, 0.7, 0.3,
  0, 0, 1), c(3,3)))

# Define code for a nimbleModel
cmp <- 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...

---

dOcc

Occupancy distribution suitable for use in codenimble 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 dOcc_s, vector for dOcc_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 (or likelihood) of observation vector x depends on occupancy probability, probOcc,
and detection probability, probDetect or probDetect[t].

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

When used directly from R, the len argument to dOcc_* 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 ex-
tension 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] ~ dOcc_s(occupancyProbability,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 dOcc_s when nimble uses the model such as for MCMC:

dOcc_s(detections[i,1:T],occupancyProbability,detectionProbability,len = T,log = 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 rOcc_s, with n = 1.

If the detection probabilities are time-dependent, use:

detections[i,1:T] ~ dOcc_v(occupancyProbability,detectionProbability[i:1:T],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

# 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
c <- 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(c, 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, 6, 13
DCJSs (dCJS), 2
DCJSv (dCJS), 2
dCJSv (dCJS), 2
dHMM, 4, 11
DHMMo, 11
dHMMo (dHMM), 4
dDynOcc, 7, 16
dDynOcc_ssm (dDynOcc), 7
dDynOcc_sss (dDynOcc), 7
dDynOcc_ssv (dDynOcc), 7
dDynOcc_ssv (dDynOcc), 7
dDynOcc_svm (dDynOcc), 7
dDynOcc_svs (dDynOcc), 7
dDynOcc_vsm (dDynOcc), 7
dDynOcc_vss (dDynOcc), 7
dDynOcc_vsv (dDynOcc), 7
dDynOcc_vvv (dDynOcc), 7
dHMM, 4, 6, 11
dHMMo, 6
dHMMo (dHMM), 11
doCC, 10, 14
doCC_s (doCC), 14
doCC_v (doCC), 14

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

rCJSss (dCJS), 2
rCJSsv (dCJS), 2
rCJSv (dCJS), 2
rCJSv (dCJS), 2
rDHMM (dHMM), 4
rDHMMo (dHMM), 4
rDynOcc_ssm (dDynOcc), 7
rDynOcc_sss (dDynOcc), 7
rDynOcc_ssv (dDynOcc), 7
rDynOcc_ssv (dDynOcc), 7
rDynOcc_vsm (dDynOcc), 7
rDynOcc_vss (dDynOcc), 7
rDynOcc_vsv (dDynOcc), 7
rDynOcc_vsv (dDynOcc), 7
rDynOcc_vvv (dDynOcc), 7
rHMM (dHMM), 11
rHMMo (dHMM), 11
rOCC_s (doCC), 14
rOCC_v (doCC), 14