Package ‘nimbleEcology’

May 4, 2021

Type Package
Title Distributions for Ecological Models in 'nimble'
Version 0.4.0
Maintainer Benjamin R. Goldstein <ben.goldstein@berkeley.edu>
Date 2021-05-03
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
VignetteBuilder knitr
RoxygenNote 7.1.1
Suggests rmarkdown, knitr, 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 2021-05-04 07:00:06 UTC
## **R topics documented:**

<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>dBetaBinom</td>
<td>2</td>
</tr>
<tr>
<td>dCJS</td>
<td>3</td>
</tr>
<tr>
<td>dDHMM</td>
<td>6</td>
</tr>
<tr>
<td>dDynOcc</td>
<td>9</td>
</tr>
<tr>
<td>dHMM</td>
<td>13</td>
</tr>
<tr>
<td>dNmixture</td>
<td>16</td>
</tr>
<tr>
<td>dOcc</td>
<td>21</td>
</tr>
<tr>
<td>nimNmixPois_logFac</td>
<td>23</td>
</tr>
</tbody>
</table>

## Index

<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>dBetaBinom</td>
<td>2</td>
</tr>
</tbody>
</table>

### Description

`dBetaBinom` and `dBetaBinom_One` provide a beta binomial distribution that can be used directly from R or in `nimble` models. These are also used by beta binomial variations of `dNmixture` distributions. `nimBetaFun` is the beta function.

### Usage

```r
nimBetaFun(a, b, log)
```

```r
dBetaBinom(x, N, shape1, shape2, log = 0)
```

```r
dBetaBinom_One(x, N, shape1, shape2, log = 0)
```

```r
rBetaBinom(n, N, shape1, shape2)
```

```r
rBetaBinom_One(n, N, shape1, shape2)
```

### Arguments

- **a**  
  shape1 argument of the beta function `nimBetaFun`.
- **b**  
  shape2 argument of the beta function `nimBetaFun`.
- **log**  
  TRUE or 1 to return log probability. FALSE or 0 to return probability.
- **x**  
  vector of integer counts.
- **N**  
  number of trials, sometimes called "size".
- **shape1**  
  shape1 parameter of the beta-binomial distribution.
- **shape2**  
  shape2 parameter of the beta-binomial distribution.
- **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`). They were originally written for the beta binomial N-mixture extensions.

The beta binomial distribution is equivalent to a binomial distribution whose probability is itself a beta distributed random variable.

The probability mass function of the beta binomial is \( \binom{N}{x} \frac{B(x + \text{shape}_1, N - x + \text{shape}_2)}{B(\text{shape}_1, \text{shape}_2)} \), where \( B(\text{shape}_1, \text{shape}_2) \) is the beta function.

The beta binomial distribution is provided in two forms. `dBetaBinom` and `rBetaBinom` are used when \( x \) is a vector (i.e. \( \text{length}(x) > 1 \)), in which case the parameters `alpha` and `beta` must also be vectors. When \( x \) is scalar, `dBetaBinom_One` and `rBetaBinom_One` are used.

Author(s)

Ben Goldstein and Perry de Valpine

See Also

For beta binomial N-mixture models, see `dNmixture`. For documentation on the beta function, use `?stats::dbeta`

Examples

```r
# Calculate a beta binomial probability
dBetaBinom(x = c(4, 0, 0, 3), N = 10,
            shape1 = c(0.5, 0.5, 0.3, 0.5), shape2 = c(0.2, 0.4, 1, 1.2))
# Same for case with one observation
dBetaBinom_One(x = 3, N = 10, shape1 = 0.5, shape2 = 0.5, log = TRUE)
```

Description

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

Usage

```r
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). Include the initial capture, so \(x[1]\) should equal 1.

- **probSurvive**
  - survival probability, either a time-independent scalar (for dCJS_s*) or a time-dependent vector (for dCJS_v*) with length \(len - 1\).

- **probCapture**
  - capture probability, either a time-independent scalar (for dCJS_*s) or a time-dependent vector (for dCJS_*v) with length \(len\). If a vector, first element is ignored, as the total probability is conditioned on the capture at \(t = 1\).

- **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.

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 (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,

```
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:

```
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

```
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**

```
# 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] ~ dCSJ_vs(probSurvive[1:4], probCapture, len = 4)
  probCapture ~ 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, checkRowSums = 1, log = 0)
dDHMMo(x, init, probObs, probTrans, len, checkRowSums = 1, log = 0)
rDHMM(n, init, probObs, probTrans, len, checkRowSums = 1)
rDHMMo(n, init, probObs, probTrans, len, checkRowSums = 1)

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 rDHMM) or time-dependent 3D array (dDHMMo and rDHMMo) 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 rDHMMo expect an additional third dimension of size (number of observation times). probObs[i, j ,(t)] is the probability that an individual in the ith latent state is recorded as being in the jth detection state (at time t). See Details for more information.
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). probTrans[i,j,t] is the probability that an individual truly in state i at time t will be in state j at time t+1. See Details for more information.
**dDHMM**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>len</td>
<td>length of observations (needed for rDHMM)</td>
</tr>
<tr>
<td>checkRowSums</td>
<td>should validity of probObs and probTrans be checked? Both of these are required to have each set of probabilities sum to 1 (over each row, or second dimension). If checkRowSums is non-zero (or TRUE), these conditions will be checked within a tolerance of 1e-6. If it is 0 (or FALSE), they will not be checked. Not checking should result in faster execution, but whether that is appreciable will be case-specific.</td>
</tr>
<tr>
<td>log</td>
<td>TRUE or 1 to return log probability. FALSE or 0 to return probability</td>
</tr>
<tr>
<td>n</td>
<td>number of random draws, each returning a vector of length len. Currently only n = 1 is supported, but the argument exists for standardization of &quot;r&quot; functions</td>
</tr>
</tbody>
</table>

**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 $S \times O$, while dDHMMo expects a three-dimensional array of time-dependent observation probabilities with dimension $S \times O \times T$, 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 $S \times S \times (T - 1)$. 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.

init has length $S$. init[i] is the probability of being in state i at the first observation time. That means that the first observations arise from the initial state probabilities.

For more explanation, see package vignette (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)],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:
rDHMM(\text{observedStates}[1:T], \text{initStates}[1:S], \text{observationProbs}[1:S,1:O], \text{transitionProbs}[1:S,1:S,1:(T-1)], 1,T, \text{log} = \text{TRUE})

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

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


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,
     0.8, 0.2), c(1, 2)))
probTrans <- array(rep(1/3, 27),  # A matrix of time-indexed transition probabilities
c(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, 1:3], len = 4, checkRowSums = 1)
    for (i in 1:3) {
    init[i] ~ dunif(0, 1)
    for (j in 1:3) {
    for (t in 1:3) {

    }  # End for (t in 1:3)
    }  # End for (j in 1:3)
}  # End for (i in 1:3)
probTrans[i,j,t] ~ dunif(0,1)
}
}

probObs[i, 1] ~ dunif(0,1)
probObs[i, 2] <- 1 - probObs[i,1]
}
}}

# 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_vvm and rDynOCC_vvm 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_vvm and rDynOCC_vvm 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)
\texttt{dDynOcc_vsv}(x, \text{init}, \text{probPersist}, \text{probColonize}, p, \text{start}, \text{end}, \log = 0) \\
\texttt{dDynOcc_svv}(x, \text{init}, \text{probPersist}, \text{probColonize}, p, \text{start}, \text{end}, \log = 0) \\
\texttt{dDynOcc_ssv}(x, \text{init}, \text{probPersist}, \text{probColonize}, p, \text{start}, \text{end}, \log = 0) \\
\texttt{rDynOcc_vvv}(n, \text{init}, \text{probPersist}, \text{probColonize}, p, \text{start}, \text{end}) \\
\texttt{rDynOcc_vsv}(n, \text{init}, \text{probPersist}, \text{probColonize}, p, \text{start}, \text{end}) \\
\texttt{rDynOcc_svv}(n, \text{init}, \text{probPersist}, \text{probColonize}, p, \text{start}, \text{end}) \\
\texttt{rDynOcc_ssv}(n, \text{init}, \text{probPersist}, \text{probColonize}, p, \text{start}, \text{end}) \\
\texttt{dDynOcc_vvs}(x, \text{init}, \text{probPersist}, \text{probColonize}, p, \text{start}, \text{end}, \log = 0) \\
\texttt{dDynOcc_vss}(x, \text{init}, \text{probPersist}, \text{probColonize}, p, \text{start}, \text{end}, \log = 0) \\
\texttt{dDynOcc_svs}(x, \text{init}, \text{probPersist}, \text{probColonize}, p, \text{start}, \text{end}, \log = 0) \\
\texttt{dDynOcc_sss}(x, \text{init}, \text{probPersist}, \text{probColonize}, p, \text{start}, \text{end}, \log = 0) \\
\texttt{rDynOcc_vvs}(n, \text{init}, \text{probPersist}, \text{probColonize}, p, \text{start}, \text{end}) \\
\texttt{rDynOcc_vss}(n, \text{init}, \text{probPersist}, \text{probColonize}, p, \text{start}, \text{end}) \\
\texttt{rDynOcc_svs}(n, \text{init}, \text{probPersist}, \text{probColonize}, p, \text{start}, \text{end}) \\
\texttt{rDynOcc_sss}(n, \text{init}, \text{probPersist}, \text{probColonize}, p, \text{start}, \text{end})

\textbf{Arguments}

\begin{itemize}
  \item \texttt{x} \hspace{1cm} \text{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.}
  \item \texttt{init} \hspace{1cm} \text{probability of occupancy in the first sampling period}
  \item \texttt{probPersist} \hspace{1cm} \text{persistence probability–probability an occupied cell remains occupied. 1-extinction probability. Scalar for \texttt{dDynOcc_s**}, vector for \texttt{dDynOcc_v**}. If vector, should have length \text{dim(x)[1]} - 1 since no transition occurs after the last observation}
  \item \texttt{probColonize} \hspace{1cm} \text{colonization probability. Probability that an unoccupied cell becomes occupied. Scalar for \texttt{dDynOcc_*s*}, vector for \texttt{dDynOcc_*v*}. If vector, should have length \text{dim(x)[1]} - 1 since no transition occurs after the last observation}
  \item \texttt{p} \hspace{1cm} \text{Detection probabilities. Scalar for \texttt{dDynOcc_*s*}, vector for \texttt{dDynOcc_*v*}, matrix for \texttt{dDynOcc_*m*}. If a matrix, dimensions should match \texttt{x}}
  \item \texttt{start} \hspace{1cm} \text{indicates the column number of the first observation in each row of \texttt{x}. A vector} \\
\end{itemize}
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 (\(\text{probPersist} \text{ or } \text{probPersist}[t-1]\)), colonization (\(\text{probColonize} \text{ or } \text{probColonize}[t-1]\)), and a detection probability (\(p, p[t], \text{ or } p[t,o]\)).

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, dDyn0cc_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 or equal to 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 (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,

\[
detections[1:T,1:O] \sim \text{dDyn0cc_ssm}(\text{init}, \text{probPersist} = \text{persistence_prob}, \text{probColonize} = \text{colonization_prob}, p = p[1:T,1:O], \text{start} = \text{start}[1:T], \text{end} = \text{end}[1:T])
\]
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_ssm, 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:T,1:O])

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(0,0,0,0,
             1,1,1,0,
             0,0,0,0,
             0,0,1,0,
             0,0,0,0), nrow = 4)
start <- c(1,1,2,1,1)
end <- c(5,5,5,4,5)
init <- 0.7
probPersist <- 0.5
probColonize <- 0.2
p <- matrix(rep(0.5, 20), nrow = 4)

# Define code for a nimbleModel
nc <- nimbleCode(
  x[1:2, 1:5] ~ dDynOcc_vvm(init,
    probPersist[1:2], probColonize[1:2], p[1:2,1:5],
    start = start[1:4], end = end[1:4])
  init ~ dunif(0,1)

# Define code for a nimbleModel
nc <- nimbleCode(
  x[1:2, 1:5] ~ dDynOcc_vvm(init,
    probPersist[1:2], probColonize[1:2], p[1:2,1:5],
    start = start[1:4], end = end[1:4])
  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 = x),
                             constants = list(start = start, end = end),
                             inits = list(p = p, probPersist = probPersist,
                                          init = init, probColonize = probColonize))

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

---

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, checkRowSums = 1, log = 0)
dHMMo(x, init, probObs, probTrans, len = 0, checkRowSums = 1, log = 0)
rHMM(n, init, probObs, probTrans, len = 0, checkRowSums = 1)
rHMMo(n, init, probObs, probTrans, len = 0, checkRowSums = 1)

**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). probObs[i, j (,t)] is the probability that an individual in the ith latent state is recorded as being in the jth detection state (at time t). See Details for more information.

probTrans
time-independent matrix of state transition probabilities. probTrans[i,j] is the probability that an individual in latent state i transitions to latent state j at the next timestep. See Details for more information.

len
length of x (see below).

checkRowSums
should validity of probObs and probTrans be checked? Both of these are required to have each set of probabilities sum to 1 (over each row, or second dimension). If checkRowSums is non-zero (or TRUE), these conditions will be checked within a tolerance of 1e-6. If it is 0 (or FALSE), they will not be checked. Not checking should result in faster execution, but whether that is appreciable will be case-specific.

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.

init has length S. init[i] is the probability of being in state i at the first observation time. That means that the first observations arise from the initial state probabilities.

For more explanation, see package vignette(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,

```
observedStates[i,1:T] ~ dHMM(initStates[1:S],observationProbs[1:S,1:O],transitionProbs[1:S,1:S],1,T)
```

declares that the `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 `dHMM` when `nimble` uses the model such as for MCMC:

```
dHMM(observedStates[1:T],initStates[1:S],observationProbs[1:S,1:O],transitionProbs[1:S,1:S],1,T,log = TRUE)
```

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 `rHMM`, with $n = 1$.

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

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

### Value

For `dHMM` and `dHMMo`: the probability (or likelihood) or log probability of observation vector $x$.

For `rHMM` and `rHMMo`: 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 `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( # 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(1, 0, 
                       0, 1, 
                       0.2, 0.8), c(2, 3)))
```
dNmixture

N-mixture distribution for use in nimble models

Description
dNmixture_s and dNmixture_v provide Poisson-Binomial mixture distributions of abundance ("N-mixture") for use in nimble models. Overdispersion alternatives are also provided.

Usage
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)
dNmixture_BNB_v(x, lambda, theta, prob, Nmin = -1, Nmax = -1, len, log = 0)
dNmixture

\begin{verbatim}
dNmixture_BNB_s(x, lambda, theta, prob, Nmin = -1, Nmax = -1, len, log = 0)
dNmixture_BNB_oneObs(x, lambda, theta, prob, Nmin = -1, Nmax = -1, len, log = 0)
dNmixture_BBP_v(x, lambda, prob, s, Nmin = -1, Nmax = -1, len, log = 0)
dNmixture_BBP_s(x, lambda, prob, s, Nmin = -1, Nmax = -1, len, log = 0)
dNmixture_BBP_oneObs(x, lambda, prob, s, Nmin = -1, Nmax = -1, len, log = 0)
dNmixture_BBNB_v(x, lambda, theta, prob, s, Nmin = -1, Nmax = -1, len, log = 0)
dNmixture_BBNB_s(x, lambda, theta, prob, s, Nmin = -1, Nmax = -1, len, log = 0)
dNmixture_BBNB_oneObs(x, lambda, theta, prob, s, Nmin = -1, Nmax = -1, len, log = 0)
rNmixture_BNB_v(n, lambda, theta, prob, Nmin = -1, Nmax = -1, len)
rNmixture_BNB_s(n, lambda, theta, prob, Nmin = -1, Nmax = -1, len)
rNmixture_BNB_oneObs(n, lambda, theta, prob, Nmin = -1, Nmax = -1, len)
rNmixture_BBP_v(n, lambda, prob, s, Nmin = -1, Nmax = -1, len)
rNmixture_BBP_s(n, lambda, prob, s, Nmin = -1, Nmax = -1, len)
rNmixture_BBP_oneObs(n, lambda, prob, s, Nmin = -1, Nmax = -1, len)
rNmixture_BBNB_v(n, lambda, theta, prob, s, Nmin = -1, Nmax = -1, len)
rNmixture_BBNB_s(n, lambda, theta, prob, s, Nmin = -1, Nmax = -1, len)
rNmixture_BBNB_oneObs(n, lambda, theta, prob, s, Nmin = -1, Nmax = -1, len)
\end{verbatim}
rNmixture_BBNB_s(n, lambda, theta, prob, s, Nmin = -1, Nmax = -1, len)

rNmixture_BBNB_oneObs(n, lambda, theta, prob, s, Nmin = -1, Nmax = -1, len)

Arguments

- **x**: vector of integer counts from a series of sampling occasions.
- **lambda**: expected value of the Poisson distribution of true abundance.
- **prob**: detection probability (scalar for \texttt{dNmixture_s}, vector for \texttt{dNmixture_v}).
- **Nmin**: minimum abundance to sum over for the mixture probability. Set to -1 to select automatically (not available for beta binomial variations; see Details).
- **Nmax**: maximum abundance to sum over for the mixture probability. Set to -1 to select automatically (not available for beta binomial variations; see Details).
- **len**: The length of the x vector.
- **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 \texttt{len}. Currently only \texttt{n = 1} is supported, but the argument exists for standardization of "r" functions.
- **theta**: abundance overdispersion parameter required for negative binomial (*NB*) N-mixture models. \(\theta\) is parameterized such that variance of the negative binomial variable \(x\) is \(\lambda^2 \times \theta + \lambda\).
- **s**: detection overdispersion parameter required for beta binomial (BB*) N-mixture models. \(s\) is parameterized such that variance of the beta binomial variable \(x\) is \(V(x) = N \times \text{prob} \times (1 - \text{prob}) \times (N + s) / (s + 1)\).

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 or negative binomial distribution. Each count, \(x[i]\) for visit \(i\), follows a binomial or beta-binomial distribution. The N-mixture distributions calculate the marginal probability of observed counts by summing over the range of latent abundance values.

The basic N-mixture model uses Poisson latent abundance with mean \(\lambda\) and binomial observed counts with size (number of trials) \(N\) and probability of success (being counted) \(\text{prob}[i]\). This distribution is available in 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.

We also provide three important variations on the traditional N-mixture model: \texttt{dNmixture_BNB}, \texttt{dNmixture_BBP}, and \texttt{dNmixture_BBNB}. These distributions allow you to replace the Poisson (P) abundance distribution with the negative binomial (NB) and the binomial (B) detection distribution with the beta binomial (BB).
NOTE: These variants should work but are considered to be in development. Their function names, parameter names, and implementations are subject to change. Use with caution while this message is present. Please contact the authors on the nimble-users listserv if you have any questions. dNmixture_v and dNmixture_s are not considered to be in development.

Binomial-negative binomial: BNB N-mixture models use a binomial distribution for detection and a negative binomial distribution for abundance with scalar overdispersion parameter \( \theta \) (0-Inf). We parameterize such that the variance of the negative binomial is \( \lambda^2 \times \theta + \lambda \), so large \( \theta \) indicates a large amount of overdispersion in abundance. The BNB is available in three suffixed forms: dNmixture_BNB_v is used if prob varies between observations, dNmixture_BNB_s is used if prob is scalar (constant across observations), and dNmixture_BNB_oneObs is used if only one observation is available at the site (so both x and prob are scalar).

Beta-binomial-Poisson: BBP N-mixture uses a beta binomial distribution for detection probabilities and a Poisson distribution for abundance. The beta binomial distribution has scalar overdispersion parameter \( s \) (0-Inf). We parameterize such that the variance of the beta binomial is \( N \times \text{prob} \times (1-\text{prob}) \times (N + s) / (s + 1) \), with greater \( s \) indicating less variance (greater-than-binomial relatedness between observations at the site) and \( s \rightarrow 0 \) indicating the binomial. The BBP is available in three suffixed forms: dNmixture_BBP_v is used if prob varies between observations, dNmixture_BBP_s is used if prob is scalar (constant across observations), and dNmixture_BBP_oneObs is used if only one observation is available at the site (so both x and prob are scalar).

Beta-binomial-negative-binomial: dNmixture_BBBN is available using a negative binomial abundance distribution and a beta binomial detection distribution. dNmixture_BBBN is available with _s, _v, and _oneObs suffixes as above and requires both arguments s and \( \theta \) as parameterized above.

The distribution dNmixture_oneObs is not provided as the probability given by the traditional N-mixture distribution for length(x) = 1 is equivalent to dpois(prob * lambda).

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

Compared to writing 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 Nmin and 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,

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

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, lambda[i] is the mean of the abundance distribution at site i, 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:

dNmixture_v(observedCounts[i,1:T],lambda[i],prob[i,1:T],Nmin,Nmax,T,log = 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:

observedCounts[1:T] ~ dNmixture_s(observedCounts[i,1:T],lambda[i],prob[i],Nmin,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) {

    
})
dOcc

\[
\text{prob[i] ~ dunif(0, 1)}
\]

# 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$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**

d0Occ_s(x, probOcc, probDetect, len = 0, log = 0)
d0Occ_v(x, probOcc, probDetect, len = 0, log = 0)
r0Occ_s(n, probOcc, probDetect, len = 0)
r0Occ_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 of observation vector \( x \) depends on occupancy probability, \( \text{probOcc} \), and detection probability, \( \text{probDetect} \) or \( \text{probDetect}[1:T] \).
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 \textit{len} argument to \texttt{dOcc_*} is not necessary. It will default to the length of \( x \). When used in \texttt{nimble} model code (via \texttt{nimbleCode}), \textit{len} must be provided (even though it may seem redundant).

For more explanation, see package vignette (\texttt{vignette("Introduction_to_nimbleEcology")}).

Compared to writing \texttt{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 \texttt{nimbleFunction}s 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 \texttt{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 \textit{log} argument.

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

\begin{verbatim}
detections[i,1:T] ~ dOcc_s(occupancyProbability,detectionProbability,T)
\end{verbatim}

declares that \( \text{detections}[i,1:T] \) (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 \texttt{dOcc_s} when \texttt{nimble} uses the model such as for MCMC:

\begin{verbatim}
dOcc_s(detections[i,1:T],occupancyProbability,detectionProbability,len = T,log = TRUE)
\end{verbatim}

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

If the detection probabilities are time-dependent, use:

\begin{verbatim}
detections[i,1:T] ~ dOcc_v(occupancyProbability,detectionProbability[1:T],len = T)
\end{verbatim}

\textbf{Value}

For \texttt{dOcc_*}: the probability (or likelihood) or log probability of observation vector \( x \).

For \texttt{rOcc_*}: a simulated detection history, \( x \).

\textbf{Author(s)}

Ben Goldstein, Perry de Valpine, and Lauren Ponisio

\textbf{See Also}

For dynamic occupancy models, see documentation for \texttt{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...
```

---

**nimNmixPois_logFac**  
*Helper function for fast N-mixture calculation*

### Description

Helper function for fast N-mixture calculation

### Usage

```r
nimNmixPois_logFac(numN, ff)
```

### Arguments

- **numN**
  - First argument for helper function nimNmixPois_logFac, representing the number of indices in the truncated sum for the N-mixture.

- **ff**
  - Second argument for helper function nimNmixPois_logFac, a derived vector of units calculated partway through the fast N-mixture algorithm.

### Details

This is a helper function for the fast N-mixture calculation. It runs an iterative calculation present in all N-mixture varieties. It doesn’t have an interpretation outside of that context.

### See Also

`dNmixture`
Index

dBetaBinom, 2
dBetaBinom_One (dBetaBinom), 2
dCJS, 3, 8, 15
dCJS_ss (dCJS), 3
dCJS_sv (dCJS), 3
dCJS_vs (dCJS), 3
dCJS_vv (dCJS), 3
dDHMM, 5, 6, 15
dDHMMo, 15
dDHMMo (dDHMM), 6
dDynOcc, 9, 20, 22
dDynOcc_ssm (dDynOcc), 9
dDynOcc_sss (dDynOcc), 9
dDynOcc_ssv (dDynOcc), 9
dDynOcc_svm (dDynOcc), 9
dDynOcc_svs (dDynOcc), 9
dDynOcc_ssv (dDynOcc), 9
dDynOcc_vsv (dDynOcc), 9
dDynOcc_vss (dDynOcc), 9
dDynOcc_vvv (dDynOcc), 9
dHMM, 5, 8, 13
dHMMo, 8
dHMMo (dHMM), 13
dNmixture, 3, 16, 23
dNmixture_BBNB_oneObs (dNmixture), 16
dNmixture_BBNB_s (dNmixture), 16
dNmixture_BBNB_v (dNmixture), 16
dNmixture_BBP_oneObs (dNmixture), 16
dNmixture_BBP_s (dNmixture), 16
dNmixture_BBP_v (dNmixture), 16
dNmixture_BNB_oneObs (dNmixture), 16
dNmixture_BNB_s (dNmixture), 16
dNmixture_BNB_v (dNmixture), 16
dNmixture_s (dNmixture), 16
dNmixture_v (dNmixture), 16
dOcc, 12, 20, 21
d0cc_s (dOcc), 21
d0cc_v (dOcc), 21

nimBetaFun (dBetaBinom), 2
nimbleCode, 3, 4, 7, 11, 14, 18, 21
nimbleModel, 3, 4, 7, 11, 14, 18, 21
nimNmixPois_logFac, 23
rBetaBinom (dBetaBinom), 2
rBetaBinom_One (dBetaBinom), 2
rCJS_ss (dCJS), 3
rCJS_sv (dCJS), 3
rCJS_vs (dCJS), 3
rCJS_vv (dCJS), 3
rDHMM (dDHMM), 6
rDHMMo (dDHMM), 6
rDynOcc_ssm (dDynOcc), 9
rDynOcc_sss (dDynOcc), 9
rDynOcc_ssv (dDynOcc), 9
rDynOcc_svm (dDynOcc), 9
rDynOcc_svs (dDynOcc), 9
rDynOcc_vsv (dDynOcc), 9
rDynOcc_vss (dDynOcc), 9
rDynOcc_vvv (dDynOcc), 9
rHMM (dHMM), 13
rHMMo (dHMM), 13

24
INDEX

rNmixture_v (dNmixture), 16
rOcc_s (dOcc), 21
rOcc_v (dOcc), 21