Journal of Statistical Software

MMMMMM YYYY, Volume VV, Issue II. *http://www.jstatsoft.org/*

unmarked: An R package for the Analysis of Wildlife Occurrence and Abundance Data

Ian J. Fiske
North Carolina State University

Richard Chandler
University of Massachusetts Amherst

Abstract

Ecological research uses data collection techniques that are prone to substantial and unique types of measurement error to address scientific questions about species abundance and distribution. These data collection schemes include a number of site-based methods: occurrence, repeated count, distance, removal, double observer sampling. To appropriately analyze these data, two-stage hierarchical models have been developed that can separately model explanatory variables of both a latent abundance process and a conditional detection process. Because these models have a straightforward interpretation paralleling how these data arose, they have recently gained immense popularity. The common two-stage structure of these models is well-suited for a unified modeling interface. The R package **unmarked** provides such an unified modeling framework, including tools for data exploration, model fitting, model criticism, post-hoc analysis, and model comparison.

Keywords: ecological, wildlife, hierarchical, occupancy, occurrence, distance, point count.

1. Introduction

1.1. Imperfect detection in ecological research

One fundamental goal of ecological research is to understand how environmental variables are related to species abundance or occurrence. Addressing these research questions is complicated by imperfect detection proability. Some animals are difficult to detect because they avoid humans, while others may have effective camouflage or simply be so rare that detections are uncommon. To overcome these difficulties, ecologists have developed a suite of robust techniques to sample animal populations. These sampling techniques include site occupancy

sampling, repeated count sampling, distance sampling, removal sampling, and double observer sampling (see Section 2 for definitions). Each of these sampling methods has some degree of imperfect detection because not all animals within a visited site are observed. Observations are generated by a combination of (1) species abundance and (2) a detection process that yields observations conditional on the abundance. Failure to account for imperfect detection can yield grossly biased estimates of abundance. Therefore, the use of these complex sampling strategies requires statistical models designed to appropriately account for the data collection process.

The scenario described above illustrates how a general unifying data generation mechanism underlies all of these site-based sampling techniques. Specifically, the general statistical model is a two-level hierarchical model with a *detection* level describing how the sampling process generates data conditional on latent or partially observed abundance, density, or occupancy rate, generally referred to as *state*. These hierarchical models composed of (1) state and (2) detection levels will be referred to as two-level ecological models (TLEMs). This paper introduces **unmarked**, an R package that provides a unified approach for estimating parameters for many popular TLEMs.

1.2. unmarked as a tool for fitting TLEMs

unmarked provides tools to assist researchers with every step of the analysis process, including data conversion and exploration, model fitting, post-hoc analysis, and model criticism. Because of the multi-level structure of the data and models, covariate data can exist at both the state and detection level. To succinctly describe these data, **unmarked** uses a new data type called the unmarkedFrame (Section 3.1). **unmarked** provides functions that import data from various common formats and convert it into **unmarked**'s data types. Once imported, **unmarked** provides functions to summarize and subset these data in a manner familiar to users of R's more common data structures such as vectors, matrices, and data frames. **unmarked** provides model-fitting functions for each of the TLEMs. The fitting functions each find the maximum likelihood estimates of parameters from a particular TLEM (Section 3.2) and return an object that can be easily manipulated. Methods exist for performing numerous post-hoc analyses such as requesting linear combinations of parameters, back-transforming parameters to constrained scales, determining confidence intervals, and evaluating goodness of fit. The model specification syntax of the fitting functions was designed in such a way that it is quite familiar to users of R's common fitting functions such as `lm` for fitting linear models.

Although there is existing software for fitting some TLEMs (e.g., **?**, for occupancy models), there are a number of advantages to a unified TLEM-fitting framework within R. Many researchers are already familiar with R and use its powerful data manipulation and plotting capabilities. Sometimes many species are analyzed in tandem, so that a common method of aggregating and post-processing of results is needed, a task easily accomplished in R. All of this is made much simpler by analyzing the data within R, and using a single environment to complete all phases of the analysis is much less error-prone than switching between applications. Another important advantage of **unmarked**'s approach is that researchers can easily simulate and analyze data within the same computational environment. This work flow permits simulation studies for power analysis calculations or the effectiveness of future sampling designs. An alternative approach for fitting TLEMs is for researchers to program their

own likelihood and maximize it within R. However, this requires that the researcher have well-developed programming abilities and also can require significant amount of overhead replicating code with minor tweaks for each new set of data. **unmarked** provides a sufficiently flexible framework that many proposed models may be fit without extensive programming by the applied researcher.

In this paper, Section 2 gives a brief summary of many of the models **unmarked** is capable of fitting. Section 3 describes general **unmarked** usage aided by a running data example.

2. Models implemented in unmarked

unmarked finds maximum likelihood estimates for the parameters of many TLEMs for popular types of site-based sampling protocols. This section provides a summary of several of these sampling techniques and how TLEMs can be used to model the resulting data. This section also introduces the syntax to illustrate how nicely it parallels the model structure, but full usage is described in Section 3.

2.1. Occurrence data

A popular estimand of interest in ecological research is the proportion of sites that are used by the study species, called occupancy rate. Another related goal is to identify factors that are associated with the changes in the probability of a site being occupied. To estimate these parameters, researchers can employ so-called occurrence sampling, whereby surveyors visit a sample of M sites and record the binary response of species detection (1) or non-detection (0) during J_i visits to the ith site during a 'season' (**?**). The key assumptions that are made when modelling these data are that the occupancy state at a site is assumed to remain constant throughout the season and repeated visits at a site are independent.

The repeated visits are necessary to gain information about the detection rate separate from the occupancy rate. To describe these data, we use the following hierarchical Bernoulli. Let ψ_i be the probability of occupancy at site i. And p_{ij} is the probability of detecting the species at site i during the jth observation given that site i is truly occupied. More formally, observations at site i arise as

$$Z_i \sim \text{Bernoulli}(\psi_i) \tag{1}$$

$$Y_{ij}|Z_i \sim \text{Bernoulli}(Z_i p_{ij}) \quad \text{for } j = 1, 2, \ldots, J_i, \tag{2}$$

where Z_i is the partially observed occupancy state.

Variables that are suspected to be related to the occupancy state are modeled as

$$\text{logit}(\psi_i) = \mathbf{x}_i' \beta, \tag{3}$$

where $\mathbf{x}_i$ is a p-vector of site-level covariates and β is a p-vector of their corresponding effect parameters. Similarly, the probability of detection can be modeled with

$$\text{logit}(p_{ij}) = \mathbf{v}_{ij}' \alpha, \tag{4}$$

where $\mathbf{v}_{ij}$ is a q-vector of observation-level covariates and α is a q-vector of their corresponding effect parameters.

Model	Fitting function	Data	Citation
Occupancy	`occu`	unmarkedFrameOccu	(?)
Royle-Nichols	`occuRN`	unmarkedFrameOccu	(?)
Point Count	`pcount`	unmarkedFramePCount	(?)
Distance-sampling	`distsamp`	unmarkedFrameDS	(?)
Arbitrary multinomial-Poisson	`multinomPois`	unmarkedFrameMPois	(?)
Colonization-extinction	`colext`	unmarkedMultFrame	(?)

Table 1: Models handled by unmarked along with their associated fitting function (Section 2) and data type (Section 3.1).

Fitting occupancy models in ***unmarked***

The above occupancy model can be fit in **unmarked** using the `occu` fitting function.

```
occu(formula, data, knownOcc, starts, method, control, se)
```

where

- `formula` is formula that specifies the covariates of detection and state in the double right-hand-sided formula: `~ v.1 + ... + v.k ~ x.1 + ... + x.l` . Standard R syntax for interactions and other formula modifications such as offsets and arbitrary functions are available. Categorical variables can be include by simply specifying factors in the formula.
- `data` is the input data as an `unmarkedFrameOccu` object.
- `knownOcc` is a numeric vector providing the indices of any sites that were known to be occupied during the sampling. These sites are sometimes included to provide extra information about the detection parameters only.
- `starts` allows the user to optionally supply starting values to the optimizer.
- `method` and `control` are directly fed into R's `optim` function, which provides the optimization engine. They allow fine control of the optimization algorithm. The default settings are to use the quasi-Newton Broyden, Fletcher, Goldfarb and Shanno (BFGS) algorithm.
- `se` is a Boolean that specifies whether or not to return a standard error computed from the Hessian.

2.2. Repeated count data

Estimates of occupancy rate provide much less information about a population than do abundance estimates. For instance, suppose the average number of individuals per site decreases from 10 to 5 during the course of a multi-season study. This dramatic population decline would not be detected using occurence sampling. One method to estimate abundance is to repeatedly visit a sample of M sites and record the number of unique individuals observed at each site. Similar assumptions are made as with occurrence data: (1) abundance at a site remains constant during a season and (2) counts at a site are independent. **?** presented the following TLEM for repeated count data. Let N_i be the unobserved total number of animals using a site and define Y_{ij} as the number of animals observed during the jth visit. Then,

$$N_i \sim F(\lambda_i, \theta) \tag{5}$$

$$Y_{ij}|N_i \sim \text{Binomial}(N_i, p_{ij}) \quad \text{for } j = 1, 2, \ldots, J_i, \tag{6}$$

where λ_i is the abundance rate at site i and p_{ij} is the detection probability during the jth visit to site i. F is a discrete distribution with support restricted to $N_i \geq 0$ and θ are extra parameters of F other than the location parameter, λ_i. **unmarked** currently supports F as Poisson or negative binomial. In the Poisson case, there is no θ. In the negative binomial case, θ is a dispersion parameter, which is useful when overdispersion is suspected.

As with the occupancy TLEM, covariates may be included in either the state (here, abundance) or detection levels, but abundance is modeled through a log link to enforce it's positivity constraint.

$$\log(\lambda_i) = \mathbf{x}_i'\beta, \tag{7}$$

where $\mathbf{x}_i$ is a p-vector of site-level covariates and β is a p-vector of their corresponding effect parameters. Similarly, the probability of detection can be modeled with

$$\text{logit}(p_{ij}) = \mathbf{v}_{ij}'\alpha, \tag{8}$$

where $\mathbf{v}_{ij}$ is a q-vector of observation-level covariates and α is a q-vector of their corresponding effect parameters.

Fitting repeated count models in ***unmarked***

The **unmarked** fitting function `pcount` estimates parameters in (5).

```
pcount(formula, data, K, mixture, starts, method, control, se)
```

where

- `formula` is formula that specifies the covariates of detection and state in the double right-hand-sided formula: `~ v.1 + ... + v.k ~ x.1 + ... + x.l` . Standard R syntax for interactions and other formula modifications such as offsets and arbitrary functions are available. Categorical variables can be include by simply specifying factors in the formula.
- `data` is the input data as an `unmarkedFramePCount` object.
- `K` is the upper index if summation used to integrate out N_i in to compute the likelihood in Equation (5).
- `mixture` specifies whether F is Poisson or negative binomial, taking values `"P"` or `"NB"` respectively.
- see Section 2.1 for details on the `starts`, `method`, and `se` arguments.

2.3. General multinomial-Poisson TLEM

Here we discuss a more general TLEM that can be modified to fit a variety of ecological sampling methods, the multinomial-Poisson model (**?**). The general form of this model is

$$N_i \sim \text{Poisson}(\lambda_i) \tag{9}$$

$$\begin{pmatrix} \mathbf{Y}_i \\ N_i - \sum_{j=1}^{J} Y_{ij} \end{pmatrix} \Bigg| N_i \sim \text{Multinomial}\left(N_i, \begin{pmatrix} \boldsymbol{\pi}_i \\ \pi_i^* \end{pmatrix}\right), \tag{10}$$

where N_i is the latent abundance at site i as with the point count model, and $\boldsymbol{\pi}_i = (\pi_{i1}, \pi_{i2}, \ldots, \pi_{iJ})'$ is the vector of cell probabilities of observing responses in the J possible categories, and $\mathbf{Y}_i$

is the J-vector of counts that were actually observed. In general, $\boldsymbol{\pi}_i$ is determined by the specific sampling method and $\sum_j \pi_{ij} \leq 1$ because detection is imperfect and $\pi_i^* = 1 - \sum_j \pi_{ij}$ is the probability of the species escaping detection at site i.

For multinomial-Poisson sampling methods, the actual observations are an underlying categorical detection variable with $R \leq J$ levels so that the J-dimensional $\mathbf{Y}_i$ is derived from the R-dimensional raw counts in some sampling method-specific manner. Thus, it is necessary to model the detection at the raw observation level, denoted p_{ik} for $k = 1, 2, \ldots, R$ at site i. Then we derive the multinomial cell probabilities $\boldsymbol{\pi}_i$ through the sampling technique-specific relationship $\pi_{ij} = g(p_{ik})$ where p_{ik} is the underlying probability of detection and g is some sampling method-specific function.

Thus, the only two requirements to adapt **unmarked**'s general multinomial Poisson modeling to a new sampling method is to specify g and a binary 0-1 matrix that describes the mapping of elements of $\mathbf{p}_i = (p_{i1}, \ldots, p_{iR})'$ to elements of $\boldsymbol{\pi}_i$. This mapping matrix, referred to in **unmarked** as `obsToY`, is necessary to consistently clean missing values from the data and relate observation-level covariates with the responses. The (j,k)th element of `obsToY` is 1 only if p_{ik} is involved in the computation of π_{ij}. The detection function g is called `piFun` in **unmarked**.

Covariates may be included in either the state (here, abundance) or detection levels, through $\mathbf{p}_i$ (not $\boldsymbol{\pi}_i$).

$$\log(\lambda_i) = \mathbf{x}_i' \beta, \tag{11}$$

where $\mathbf{x}_i$ is a p-vector of site-level covariates and β is a p-vector of their corresponding effect parameters. Similarly, the probability of detection can be modeled with

$$\text{logit}(p_{ij}) = \mathbf{v}_{ij}' \alpha, \tag{12}$$

where $\mathbf{v}_{ij}$ is a q-vector of observation-level covariates and α is a q-vector of their corresponding effect parameters.

*Fitting multinomial-Poisson models in **unmarked***

The `multinomPois` fitting function is available in **unmarked** for fitting multinomial-Poisson TLEMs.

```
multinomPois(formula, data, starts, method, control, se)
```

where

- `formula` is formula that specifies the covariates of detection and state in the double right-hand-sided formula: `~ v.1 + ... + v.k ~ x.1 + ... + x.l` . Standard R syntax for interactions and other formula modifications such as offsets and arbitrary functions are available. Categorical variables can be include by simply specifying factors in the formula.
- `data` is the input data as an `unmarkedFrameMPois` object.
- see Section 2.1 for details on the `starts`, `method`, and `se` arguments.

We now describe two common sampling methods that can be modeled with the multinomial-Poisson model: removal sampling and double observer sampling. These two methods are included in **unmarked**, but additional methods may easily be specified by the user with only the `piFun` function and `obsToY` matrix.

Removal sampling

Popular in fisheries, removal sampling is implemented by visiting a sample of M sites J times each and trapping and removing individuals at each visit with the same effort. Thus, Y_{ij} is the number of individuals captured at the jth visit for $j = 1, 2, \ldots, J$.

Then, we can specify g for removal sampling as follows. The probability of an individual at site i being captured on the first visit is $\pi_{i1} = p_{i1}$. The probability of capture on the jth visit is

$$\pi_{ij} = \prod_{k=1}^{j-1}(1 - p_{ik})p_{ij}, \tag{13}$$

for $j = 2, \ldots, J$ and the probability of not being sampled is

$$\pi_{i,J+1} = \prod_{j=1}^{J}(1 - p_{ij}) \tag{14}$$

Or, equivalently,

$$\pi_i = \begin{pmatrix} p_{i1} \\ (1 - p_{i1})p_{i2} \\ \vdots \\ \prod_{j=1}^{J}(1 - p_{ij})p_{iJ} \end{pmatrix} \tag{15}$$

Thus, the mapping matrix is an $J \times R$ upper triangular matrix with ones in the upper triangle,

$$\begin{pmatrix} 1 & 1 & \ldots & 1 \\ 0 & 1 & \ldots & 1 \\ \vdots & & \ddots & \vdots \\ 0 & 0 & \ldots & 1 \end{pmatrix} \tag{16}$$

Double observer sampling

Double observer sampling collects data by a team of two surveyors simultaneously visiting a site. They each collect lists of identified animals. Thus the counts in the three cells of $\mathbf{Y}_i$ correspond to the numbers of individuals seen by observer one, observer two, or both. Thus, for double observer sampling, g is defined as follows.

$$\pi_i = \begin{pmatrix} p_{i1}(1 - p_{i2}) \\ (1 - p_{i1})p_{i2} \\ p_{i1}p_{i2} \\ (1 - p_{i1})(1 - p_{i2}). \end{pmatrix} \tag{17}$$

The `obsToY` mapping matrix for double observer sampling is the following 3×2 matrix.

$$\begin{pmatrix} 1 & 0 \\ 0 & 1 \\ 1 & 1 \end{pmatrix} \tag{18}$$

2.4. Distance sampling

In distance sampling, surveyors sample sites by traveling along a transect or standing at a so-called point transect and recording the distance to each detected study animal (**?**). For the model of (**?**) used in **unmarked** the continuous distances to each observed animal must be binned into categories according to cut-points $c_1, c_2, \ldots, c_J$. Then Y_{ij} is the count of observations at site i falling into the jth distance bin. The user must specify a detection function $g(x;\theta)$ that describes the probability of observing an animal as a function of distance x from the transect. The cell probabilities π_i are constructed by integrating g over the distance classes.

As currently implemented, the general form of the distance sampling model is identical to the multinomial-Poisson mixture described above, with the sole difference being that instead of

$$\text{logit}(p_{ij}) = \mathbf{v}'_{ij}\alpha, \tag{19}$$

we have

$$\log(\theta_i) = \mathbf{v}'_i\alpha, \tag{20}$$

Here, the log link is required because θ is a positive shape parameter of the detection function. Eventually, alternative models (*eg* the negative binomial) of the latent abundance variable may be added.

Fitting distance sampling models in ***unmarked***

The above distance model can be fit in **unmarked** using the `distsamp` fitting function.

```
distamp(formula, data, keyfun = c("halfnorm", "exp", "hazard",
    "uniform"), output = c("density", "abund"), unitsOut = c("ha",
    "kmsq"), starts = NULL, method = "BFGS", control = list(),
    se = TRUE)
```

- `formula` is formula that specifies the covariates of detection and state in the double right-hand-sided formula: `~ v.1 + ... + v.k ~ x.1 + ... + x.l` . Standard R syntax for interactions and other formula modifications such as offsets and arbitrary functions are available. Categorical variables can be include by simply specifying factors in the formula.
- `data` is the input data as an `unmarkedFrameDS` object.
- `keyfun` specifies the detection function.
- `output` determines how the response is modeled when all transect lengths and distance intervals are equal. Otherwise, the matrix of site by distance interval areas must be accounted for and density is modeled.
- `unitsOut` the output units when modeling density.
- see Section 2.1 for details on the `starts`, `method`, and `se` arguments.

2.5. Colonization-extinction data

Sometimes the study objective is to understand the evolution of occupancy state over time. To obtain such information researchers conduct repeated occupancy studies (see Section 2.1) to the same sample of sites over consecutive seasons (**?**). They then seek to estimate probabilities of colonization (γ_{it}) and extinction (ϵ_{it}), where colonization is the change of an unoccupied site to occupied and extinction if the change of an occupied site to unoccupied. If the occupancy status is assumed to evolve according to a Markov process, then a 2-state finite hidden Markov model describes these data. Let Y_{itj} denote the observed animal occurrence status at visit j during season t to site i. Then

$$Z_{i1} \sim \text{Bernoulli}(\psi) \tag{21}$$

$$Z_{it} \sim \begin{cases} \text{Bernoulli}(\gamma_{i(t-1)}) & \text{if } Z_{i(t-1)} = 0 \\ \text{Bernoulli}(1 - \epsilon_{i(t-1)}) & \text{if } Z_{i(t-1)} = 1 \end{cases}, \tag{22}$$

$$\text{for } t = 2, 3, \ldots, T$$

$$Y_{itj} | Z_{it} \sim \text{Bernoulli}(Z_{it} p_{itj}) \tag{23}$$

Fitting colonization-extinction models in ***unmarked***

Colonization-extinction models in **unmarked** are fit with the `colext` fitting function.

```
colext(psiformula, gammaformula, epsilonformula, pformula, data,
starts, method, control, se)
```

- `psiformula` is a right-hand sided formula for the initial probability of occupancy (ψ) at each site.
- `gammaformula` is a right-hand sided formula for colonization probability (γ).
- `epsilonformula` is a right-hand sided formula for extinction probability (ϵ).
- `pformula` is a right-hand sided formula for detection probability.
- `data` is an unmarkedMultFrame object that supplies the data.
- see Section 2.1 for details on the `starts`, `method`, and `se` arguments.

3. unmarked usage

By exploiting the common structure of TLEMs, **unmarked** is able to provide data structures, fitting syntax, and post-processing that form a cohesive framework for site-based ecological data analysis. In order to achieve these goals, **unmarked** uses the S4 class system (**?**). As R's most modern system of class-based programming, S4 allows customization of functions, referred to as methods, to specific object classes and superclasses. For example, when the generic `predict` method is called with any **unmarked** model fit object as an argument, the

actual `predict` implementation depends on the specific TLEM that was fit. Use of class-based programming can provide more reliable and maintainable software while also making the program more user-friendly (**?**).

3.1. Preparing data

unmarked uses a custom S4 data structure called the unmarkedFrame to store all data related to a sampling survey. Although this at first appears to add an extra layer of work for the user, there are several reasons for this design choice. The multilevel structure of TLEMs means that standard rectangular data structures such as data frames or matrices are not suitable for storing the data. For example, covariates might have been measured separately at the site level and at the visit level. Furthermore, the length of the response vector $\mathbf{Y}_i$ at site i might differ from the number of observations at the site as in the multinomial Poisson model. Aside from these technical reasons, **?** pointed out that the use of such portable custom data objects can simplify future reference to previous analyses, an often neglected aspect of research. Repeated fitting calls using the same set of data require less code repetition if all data is contained in a single object. Finally, calls to fitting functions have a cleaner appearance with a more obvious purpose when the call is not buried in data arguments. The parent data class is called an unmarkedFrame and each **unmarked** fitting function has its own data type that extends the unmarkedFrame. Thus, the first step when using **unmarked** is to import data into the proper type of unmarkedFrame. To ease this step, **unmarked** includes several helper functions to automatically convert data into an unmarkedFrame: `csvToUMF` which imports data directly from a comma-separated value text file, `formatWide` and `formatLong` which convert data from data frames, and the family of unmarkedFrame constructor functions.

An unmarkedFrame object contains slots for the observation matrix `y`, a data frame of site-level covariates `siteCovs`, and a data frame of observation-level covariates `obsCovs`. The `y` matrix is the only required unmarkedFrame slot. `y` is an $M \times J$ matrix with the ith row containing the J-dimensional $\mathbf{y}_i$. If $J_i \leq J$, then any number of elements of $\mathbf{y}_i$ can take the value `NA` to denote this. `siteCovs` is an M-row data frame with a column for each site-level covariate. `obsCovs` is an MJ-row data frame with a column for each observation-level covariate. Thus each row of `obsCovs` corresponds to a particular observation, with the order corresponding to site varying slower and observation within site varying faster. Both `siteCovs` and `obsCovs` can contain `NA` values corresponding to unbalanced data collection. If any terms in the model specification are missing for that observation, **unmarked** automatically removes observations. **unmarked** provides constructor functions to make creating unmarkedFrames straightforward. For each specific data type, specific types of unmarkedFrames extend the basic unmarkedFrame to handle model-specific nuances.

Importing repeated count data

Here is an example of creating an unmarkedFrame for point count data (Section 2.2). from a list a matrix of observations, a data frame of site-level covariates, and a list of matrices of observation-level covariates.

First, load the included data set of Mallard duck point counts from **?** and visually inspect the point count matrix, which is already in the same format as the `y` slot of an unmarkedFrame.

```
> library(unmarked)
```

```
> data(mallard)
> head(mallard.y)

     y.1 y.2 y.3
[1,]   0   0   0
[2,]   0   0   0
[3,]   3   2   1
[4,]   0   0   0
[5,]   3   0   3
[6,]   0   0   0

> dim(mallard.y)

[1] 239   3
```

So the data have 239 sites and a maximum of 3 observations per site.

In this example, site-level covariates are already in the same data frame format as the `siteCovs` slot.

```
> summary(mallard.site)

      elev                length              forest
 Min.   :-1.436e+00   Min.   :-4.945e+00   Min.   :-1.265e+00
 1st Qu.:-9.565e-01   1st Qu.:-5.630e-01   1st Qu.:-9.560e-01
 Median :-1.980e-01   Median : 4.500e-02   Median :-6.500e-02
 Mean   :-4.603e-05   Mean   :-2.929e-05   Mean   : 6.695e-05
 3rd Qu.: 9.940e-01   3rd Qu.: 6.260e-01   3rd Qu.: 7.900e-01
 Max.   : 2.434e+00   Max.   : 2.255e+00   Max.   : 2.299e+00
```

However, observation level covariates are a list object with separate $M \times J$ matrices for each observation-level covariate. The unmarkedFrame constructors can accept `obsCovs` in this list format or as a data frame in the format described above.

```
> str(mallard.obs)

List of 2
 $ ivel: num [1:239, 1:3] -0.506 -0.934 -1.136 -0.819 0.638 ...
  ..- attr(*, "dimnames")=List of 2
  .. ..$ : NULL
  .. ..$ : chr [1:3] "ivel.1" "ivel.2" "ivel.3"
 $ date: num [1:239, 1:3] -1.76 -2.9 -1.69 -2.19 -1.83 ...
  ..- attr(*, "dimnames")=List of 2
  .. ..$ : NULL
  .. ..$ : chr [1:3] "date.1" "date.2" "date.3"
```

The following call to `unmarkedFramePCount` function creates an unmarkedFramePCount object, which is the specific type of unmarkedFrame for point count data.

```
> mallardUMF <- unmarkedFramePCount(y = mallard.y, siteCovs = mallard.site,
+     obsCovs = mallard.obs)
```

Printing unmarkedFrames shows them as data frames to make it easier to verify that the data was converted correctly.

```
> head(mallardUMF)

Data frame representation of unmarkedFrame object.
   y.1 y.2 y.3   elev length forest ivel.1 ivel.2 ivel.3 date.1
1    0   0   0 -1.173  0.801 -1.156 -0.506 -0.506 -0.506 -1.761
2    0   0   0 -1.127  0.115 -0.501 -0.934 -0.991 -1.162 -2.904
3    3   2   1 -0.198 -0.479 -0.101 -1.136 -1.339 -1.610 -1.690
4    0   0   0 -0.105  0.315  0.008 -0.819 -0.927 -1.197 -2.190
5    3   0   3 -1.034 -1.102 -1.193  0.638  0.880  1.042 -1.833
6    0   0   0 -0.848  0.741  0.917 -1.329 -1.042 -0.899 -2.619
7    0   0   0 -0.910  0.115 -1.083 -1.448 -1.562 -1.676 -2.690
8    0   0   0 -1.003 -1.007 -0.792 -0.321 -0.557 -0.636 -2.119
9    0   0   0 -0.058 -0.913  0.553 -0.231 -0.231 -0.001 -2.047
10   0   0   0 -0.631  1.556  0.808 -1.097 -1.021 -0.832 -2.333
   date.2 date.3
1   0.310  1.381
2  -1.047  0.596
3  -0.476  1.453
4  -0.690  1.239
5   0.167  1.381
6   0.167  1.381
7  -1.190  1.596
8  -0.476  1.453
9  -0.547  1.167
10 -1.119 -0.261
```

We can also check data contents with a quick summary.

```
> summary(mallardUMF)

unmarkedFrame Object

239 sites
Maximum number of observations per site: 3
Mean number of observations per site: 2.76
Sites with at least one detection: 40

Tabulation of y observations:
   0    1    2    3    4    7   10   12 <NA>
 576   54   11    9    6    1    1    1   58
```

```
Site-level covariates:
      elev                length              forest
 Min.   :-1.436e+00   Min.   :-4.945e+00   Min.   :-1.265e+00
 1st Qu.:-9.565e-01   1st Qu.:-5.630e-01   1st Qu.:-9.560e-01
 Median :-1.980e-01   Median : 4.500e-02   Median :-6.500e-02
 Mean   :-4.603e-05   Mean   :-2.929e-05   Mean   : 6.695e-05
 3rd Qu.: 9.940e-01   3rd Qu.: 6.260e-01   3rd Qu.: 7.900e-01
 Max.   : 2.434e+00   Max.   : 2.255e+00   Max.   : 2.299e+00

Observation-level covariates:
      ivel                 date
 Min.   :-1.753e+00   Min.   :-2.904e+00
 1st Qu.:-6.660e-01   1st Qu.:-1.119e+00
 Median :-1.390e-01   Median :-1.190e-01
 Mean   : 1.504e-05   Mean   : 7.259e-05
 3rd Qu.: 5.490e-01   3rd Qu.: 1.310e+00
 Max.   : 5.980e+00   Max.   : 3.810e+00
 NA's   : 5.200e+01   NA's   : 4.200e+01
```

The summary reveals that only 40 sites have at least one detection – these data are in fact very sparse as is commonly place in ecological statistics. Notice that these data already appear centered and scaled. This is common practice to improve numerical stability of the estimation procedure.

Importing removal sampling data

To illustrate the slightly different syntax for removal sampling data example, we will import removal data from a removal survey of Ovenbird described by **?**. The data consist of a list with a matrix `data` containing the removal counts for 4 visits and a data frame called `covariates` containing site-level covariates information.

```
> data(ovendata)
> str(ovendata.list)

List of 2
 $ data      : num [1:70, 1:4] 0 1 0 0 0 0 0 0 2 1 ...
 $ covariates:'data.frame':        70 obs. of  3 variables:
  ..$ site: Factor w/ 70 levels "CAT003","CAT004",..: 1 2 3 4 5 6 7 8 9 10 ...
  ..$ ufp : num [1:70] 23.4 15.6 35.5 25.8 25 ...
  ..$ trba: num [1:70] 62.5 70 90 60 122.5 ...

> ovendata.list$data[sample(1:70, 10), ]

      [,1] [,2] [,3] [,4]
 [1,]    0    1    0    0
 [2,]    0    0    0    0
```

```
 [3,]    2    0    0    0
 [4,]    3    0    0    0
 [5,]    2    0    1    0
 [6,]    0    0    0    0
 [7,]    0    0    0    0
 [8,]    1    1    0    0
 [9,]    1    0    0    0
[10,]    0    0    0    0
```

```
> ovendata.list$covariates[sample(1:70, 10), ]
```

```
     site   ufp  trba
11 CAT068  9.37  85.0
67 FCW441  8.20 100.0
6  CAT023 23.43 107.5
34 CAT293  4.29 155.0
46 FCW131 17.18  77.5
35 CAT346  5.85 135.0
42 FCW085 25.39  80.0
20 CAT129  9.76 145.0
31 CAT264 24.21 117.5
60 FCW299 16.01  95.0
```

The only additional specification when importing removal data is to specify the particular type of multinomial-Poisson data as `removal` via the `type` argument.

```
> scaled.covs <- as.data.frame(scale(ovendata.list$covariates[,
+     -1]))
> ovenFrame <- unmarkedFrameMPois(ovendata.list$data, siteCovs = scaled.covs,
+     type = "removal")
> summary(ovenFrame)
```

```
unmarkedFrame Object

70 sites
Maximum number of observations per site: 4
Mean number of observations per site: 4
Sites with at least one detection: 44

Tabulation of y observations:
   0    1    2    3 <NA>
 218   49   11    2    0

Site-level covariates:
      ufp                 trba
 Min.   :-1.471e+00   Min.   :-2.010e+00
```

```
 1st Qu.:-7.408e-01   1st Qu.:-6.931e-01
 Median :-2.535e-01   Median :-1.287e-01
 Mean   : 1.351e-17   Mean   :-2.531e-16
 3rd Qu.: 9.844e-01   3rd Qu.: 7.178e-01
 Max.   : 2.344e+00   Max.   : 2.881e+00
```

3.2. Fitting TLEMs

As introduced in Section 2, each TLEM has a corresponding fitting function. For example, to fit a repeated count model, we call `pcount`. Table 1 provides a summary of all models that **unmarked** currently fits. With the exception of colext, all fitting functions share a double right-hand sided formula syntax that expresses the hierarchical model and data structures. No response side of the formula is specified because the unmarkedFrame defines the response variable uniquely as the `y` slot. The first component of the formula describes the observation-level model of the detection process and the second component describes the site-level model of the state process.

Fitting a repeated count model

Continuing the Mallard example, the following call to `pcount` fits an binomaial-Poisson mixture model (Section 2.2). Here, we specify that the detection probability p should be modeled by day of year, including a quadratic term. We also wish to model abundance using elevation and proportion of area forested. As described in Section 2.2, covariates of detection are on the logit-scale and covariates of abundance are on the log scale for the point count model.

```
> fm.mallard.1 <- pcount(~date + I(date^2) ~ elev + forest,
+     data = mallardUMF)
> summary(fm.mallard.1)

Call:
pcount(formula = ~date + I(date^2) ~ elev + forest, data = mallardUMF)

Abundance (log-scale):
            Estimate    SE     z  P(>|z|)
(Intercept)   -1.853 0.236 -7.87 3.62e-15
elev          -1.232 0.234 -5.27 1.36e-07
forest        -0.756 0.165 -4.60 4.32e-06

Detection (logit-scale):
            Estimate     SE       z P(>|z|)
(Intercept)   0.3071 0.2149  1.4290 0.15301
date         -0.4091 0.1482 -2.7612 0.00576
I(date^2)    -0.0077 0.0853 -0.0903 0.92801

AIC: 521.2001
Number of sites: 235
Sites removed: 12 69 118 146
```

```
optim convergence code: 0
optim iterations: 37
Bootstrap iterations: 0
```

This initial fit suggests that Mallard abundance decreases with increasing elevation and forest. It also looks like a linear model might suffice for the detection model, so we subsequently fit the linear detection model.

```
> fm.mallard.2 <- pcount(~date ~ elev + forest, data = mallardUMF)
> fm.mallard.2

Call:
pcount(formula = ~date ~ elev + forest, data = mallardUMF)

Abundance:
            Estimate    SE     z  P(>|z|)
(Intercept)   -1.857 0.232 -7.99 1.37e-15
elev          -1.236 0.230 -5.38 7.59e-08
forest        -0.756 0.164 -4.60 4.23e-06

Detection:
            Estimate    SE     z  P(>|z|)
(Intercept)    0.298 0.188  1.58 0.113983
date          -0.401 0.114 -3.51 0.000449

AIC: 519.2081
```

This seems to be a better model according to both the Wald p-value and AIC. Thus detection appears to decrease during the course of a year.

Fitting a multinomial-Poisson model

Here we demonstrate fitting a multinomial-Poisson model to removal sampling data. The Ovenbird data has no visit-level covariates, so detection probability is assumed constant across visits. It is not necessary to specify that removal sampling was used when fitting the model because this information is already stored in the `ovenFrame` data. We model the abundance with understory forest coverage (`ufp`) and average basal tree area (`trba`).

```
> fm.oven.1 <- multinomPois(~1 ~ ufp + trba, ovenFrame)
> fm.oven.1

Call:
multinomPois(formula = ~1 ~ ufp + trba, data = ovenFrame)

Abundance:
            Estimate    SE      z P(>|z|)
(Intercept)    0.102 0.119  0.864   0.388
```

```
ufp            0.100 0.126  0.794   0.427
trba          -0.171 0.135 -1.262   0.207

Detection:
 Estimate    SE    z P(>|z|)
    0.288 0.233 1.24   0.217

AIC: 326.1387
```

3.3. Examining model fits

Objects returned by **unmarked**'s fitting functions also make use of the S4 class system. All model fit objects belong to the unmarkedFit parent class. Thus, common operations such as extracting coefficient estimates, covariance matrices for estimates, and confidence intervals have been adapted to behave similar to R's base functions.

For example, we can extract estimated coefficients either from the entire model, or from the state or detection levels by specifying the `type` argument.

```
> coef(fm.mallard.2)

   lam(Int)   lam(elev) lam(forest)      p(Int)     p(date)
 -1.8565048  -1.2355497  -0.7555178   0.2977558  -0.4006066

> coef(fm.mallard.2, type = "state")

   lam(Int)   lam(elev) lam(forest)
 -1.8565048  -1.2355497  -0.7555178
```

To check which types are available for a model, use the `names` method.

```
> names(fm.mallard.2)

[1] "state" "det"
```

Similarly, the `vcov` function extracts the covariance matrix of the estimates, using the observed Fisher information by default.

```
> vcov(fm.mallard.2)

                lam(Int)    lam(elev)   lam(forest)       p(Int)
lam(Int)     0.054013257  0.040740133  0.0080777484 -0.006188545
lam(elev)    0.040740133  0.052810074 -0.0084624468 -0.001751201
lam(forest)  0.008077748 -0.008462447  0.0269780875  0.001779971
p(Int)      -0.006188545 -0.001751201  0.0017799714  0.035490236
p(date)     -0.001785422 -0.003393574  0.0005126907  0.003448530
```

```
                 p(date)
lam(Int)    -0.0017854217
lam(elev)   -0.0033935742
lam(forest)  0.0005126907
p(Int)       0.0034485301
p(date)      0.0130286462
```

vcov also accepts a `type` argument.

```
> vcov(fm.mallard.2, type = "state")

               lam(Int)    lam(elev)  lam(forest)
lam(Int)    0.054013257  0.040740133  0.008077748
lam(elev)   0.040740133  0.052810074 -0.008462447
lam(forest) 0.008077748 -0.008462447  0.026978087
```

unmarked also provides an associated helper function `SE` to return standard errors from the square root of the diagonal of these covariance matrices.

```
> SE(fm.mallard.2)

   lam(Int)   lam(elev) lam(forest)      p(Int)     p(date)
  0.2324075   0.2298044   0.1642501   0.1883885   0.1141431

> SE(fm.mallard.2, type = "state")

   lam(Int)   lam(elev) lam(forest)
  0.2324075   0.2298044   0.1642501
```

Nonparametric bootstrapping can also be used to estimate the covariance matrix. **unmarked** implements a two-stage bootstrap in which the sites are first drawn with replacement, and then within each site, the observations are drawn with replacement. First, bootstrap draws must be taken using the `nonparboot`. `nonparboot` returns a new version of the unmarkedFit object with additional bootstrap sampling information. Thus, this new fit must be stored, either in a new fit object or the same one, and then subsequently queried for bootstrap summaries. In the following, bootstrapping can be quite slow. For these examples, we illustrate with the removal sampling data simply because computations are much faster. However, bootstrapping is available for any of the TLEMs that **unmarked** fits.

```
> set.seed(1234)
> fm.oven.1 <- nonparboot(fm.oven.1, B = 100)

> SE(fm.oven.1, type = "state")

 lambda(Int)  lambda(ufp) lambda(trba)
   0.1185431    0.1262615    0.1353444
```

```
> SE(fm.oven.1, type = "state", method = "nonparboot")

 lambda(Int)  lambda(ufp) lambda(trba)
   0.1256949    0.1156153    0.1197142
```

It looks like bootstrapping and asymptotic standard errors are close. The summary now states the number of bootstrap samples.

```
> summary(fm.oven.1)

Call:
multinomPois(formula = ~1 ~ ufp + trba, data = ovenFrame)

Abundance (log-scale):
            Estimate    SE      z P(>|z|)
(Intercept)    0.102 0.119  0.864   0.388
ufp            0.100 0.126  0.794   0.427
trba          -0.171 0.135 -1.262   0.207

Detection (logit-scale):
 Estimate    SE    z P(>|z|)
    0.288 0.233 1.24   0.217

AIC: 326.1387
Number of sites: 70
optim convergence code: 0
optim iterations: 33
Bootstrap iterations: 100
```

Additional bootstrap samples can be drawn by calling `nonparboot` again.

```
> fm.oven.1 <- nonparboot(fm.oven.1, B = 100)
> summary(fm.oven.1)

Call:
multinomPois(formula = ~1 ~ ufp + trba, data = ovenFrame)

Abundance (log-scale):
            Estimate    SE      z P(>|z|)
(Intercept)    0.102 0.119  0.864   0.388
ufp            0.100 0.126  0.794   0.427
trba          -0.171 0.135 -1.262   0.207

Detection (logit-scale):
 Estimate    SE    z P(>|z|)
    0.288 0.233 1.24   0.217
```

```
AIC: 326.1387
Number of sites: 70
optim convergence code: 0
optim iterations: 33
Bootstrap iterations: 200
```

Confidence intervals can be requested for the coefficients at either stage of the model. By default, the asymptotic normal approximation is used.

```
> confint(fm.oven.1, type = "state", level = 0.95)

                  0.025      0.975
lambda(Int)  -0.1299722 0.33470834
lambda(ufp)  -0.1471741 0.34776195
lambda(trba) -0.4361311 0.09440937
```

Profile confidence intervals are also available upon request. This can take some time, however, because for each parameter, a nested optimization within a root-finding algorithm is being used to find the profile limit.

```
> ci <- confint(fm.oven.1, type = "state", level = 0.95,
+     method = "profile")

Profiling parameter 1 of 3 ... done.
Profiling parameter 2 of 3 ... done.
Profiling parameter 3 of 3 ... done.

> ci

                  0.025      0.975
lambda(Int)  -0.1390786 0.32676614
lambda(ufp)  -0.1477724 0.34811770
lambda(trba) -0.4368444 0.09469605
```

The profile confidence intervals and normal approximations are quite similar here.

Linear combinations of estimates

Often, meaningful hypotheses can be addressed by estimating linear combinations of coefficient estimates. Linear combinations of coefficient estimates can be requested with `linearComb`.

Continuing the Ovenbird example, the following code estimates the log-abundance rate for a site with `ufp = 0.5` and `trba = 0`.

```
> (lc <- linearComb(fm.oven.1, type = "state", coefficients = c(1,
+     0.5, 0)))
```

```
Linear combination(s) of Abundance estimate(s)

 Estimate    SE (Intercept) ufp trba
    0.153 0.130           1 0.5    0
```

Multiple sets of coefficients may be supplied as a matrix. The following code requests the estimated log-abundance for sites with `ufp = 0.5` and `trba = 1`.

```
> (lc <- linearComb(fm.oven.1, type = "state", coefficients = matrix(c(1,
+     0.5, 0, 1, 1, 0), 2, 3, byrow = TRUE)))

Linear combination(s) of Abundance estimate(s)

  Estimate    SE (Intercept) ufp trba
1    0.153 0.130           1 0.5    0
2    0.203 0.166           1 1.0    0
```

Standard errors and confidence intervals are also available for linear combinations of parameters. By requesting parametric bootstrapped standard errors, **unmarked** uses the samples that were drawn earlier.

```
> SE(lc)

[1] 0.1296526 0.1659460

> SE(lc, method = "nonparboot")

[1] 0.1354092 0.1659877

> confint(lc)

       0.025     0.975
1 -0.1015993 0.4066294
2 -0.1225862 0.5279102
```

Back-transforming linear combinations of coefficients

Estimates of linear combination back-transformed to the native scale are likely to be more interesting than the direct linear combinations. For example, the logistic transformation is applied to estimates of detection rates, resulting in a probability bound between 0 and 1. This is accomplished with the `backTransform`. Standard errors of back-transformed estimates are estimated using the delta method. Confidence intervals are estimated by back-transforming the confidence interval of the original linear combination.

```
> (btlc <- backTransform(lc))
```

```
Backtransformed linear combination(s) of Abundance estimate(s)

  Estimate    SE LinComb (Intercept) ufp trba
1     1.16 0.151   0.153           1 0.5    0
2     1.22 0.203   0.203           1 1.0    0

Transformation: exp
```

```
> SE(btlc)

[1] 0.1510141 0.2032272

> SE(btlc, method = "nonparboot")

[1] 0.1577193 0.2032782

> confint(btlc)

      0.025    0.975
1 0.9033915 1.501747
2 0.8846296 1.695386
```

Model selection

unmarked performs simple AIC-based model selection for structured lists of unmarkedFit objects. Implementing alternative methods of variable selection for these models is an area of future research. To demonstrate, we fit a few more models to the Ovenbird removal data, including all an interaction model, two models with single predictors, and a null model with no predictors.

```
> fm.oven.2 <- update(fm.oven.1, formula = ~1 ~ ufp * trba)
> fm.oven.3 <- update(fm.oven.1, formula = ~1 ~ ufp)
> fm.oven.4 <- update(fm.oven.1, formula = ~1 ~ trba)
> fm.oven.5 <- update(fm.oven.1, formula = ~1 ~ 1)
```

Now, we can organize the fitted models with the `fitList` function and the use the `modSel` method to rank the models by AIC and print other relevant information such as Nagelkerke's (**?**) R^2.

```
> fmList <- fitList(Global = fm.oven.2, additive = fm.oven.1,
+     ufp = fm.oven.3, trba = fm.oven.4, Null = fm.oven.5)
> modSel(fmList, nullmod = "Null")

          n nPars    AIC delta AICwt cumltvWt   Rsq
trba     70     3 324.77  0.00  0.35     0.35 0.049
ufp      70     3 325.73  0.96  0.21     0.56 0.036
additive 70     4 326.14  1.37  0.17     0.73 0.058
Null     70     2 326.28  1.51  0.16     0.90 0.000
Global   70     5 327.17  2.40  0.10     1.00 0.071
```

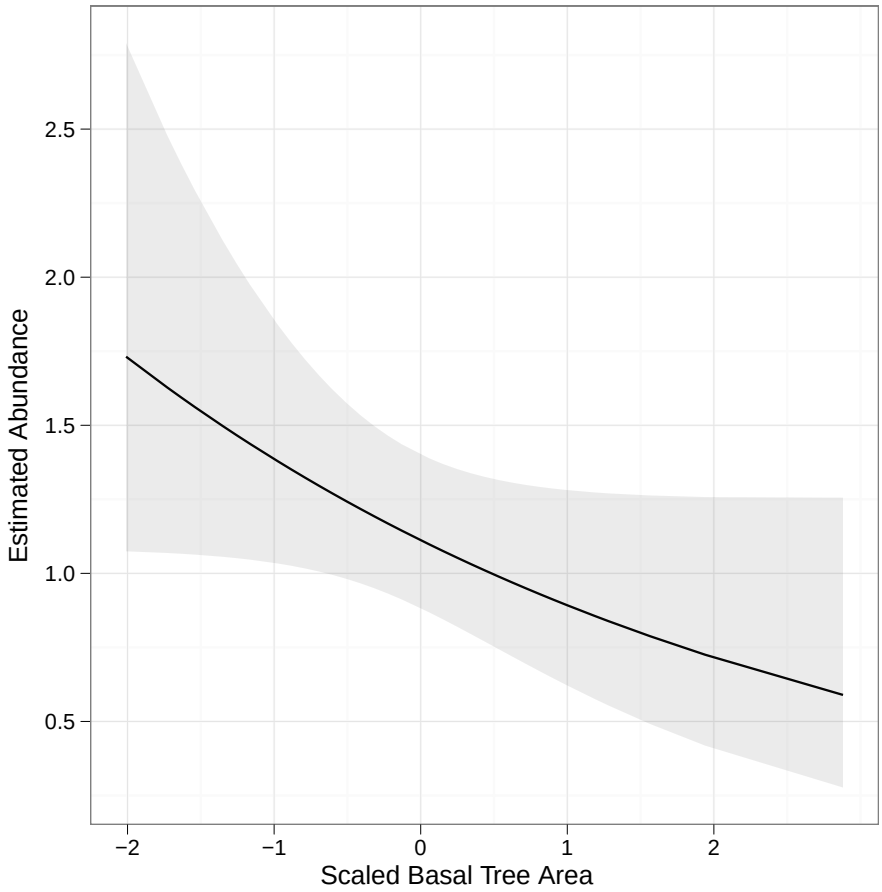

Figure 1: Examine estimated abundance for Ovenbird removal data. Band is 95% confidence interval.

It looks like the best model includes only tree basal area as a predictor of abundance. As **?** pointed out, this is a reasonable result because tree basal area and understory forest coverage are negatively correlated. We can examine the relationship between basal tree area and estimated abundance with the `predict` method (Figure 1).

```
> preddata <- predict(fm3, type = "state", appendData = TRUE)
> library(ggplot2)
> qplot(trba, Predicted, data = preddata, geom = "line",
+     xlab = "Scaled Basal Tree Area", ylab = "Estimated Abundance") +
+     geom_ribbon(aes(x = trba, ymin = lower, ymax = upper),
+         alpha = 0.1) + theme_bw()
```

Parametric bootstrap for goodness of fit

AIC can be used to select the best model in a set, but it does not indicate how well a model fits the data. To conduct goodness of fit tests, **unmarked** provides a generic parametric

bootstrapping function. It simulated data from the fitted model and the computes summary information from any specified statistic, with the default being the sum of squared residuals.

```
> set.seed(1234)
> pb <- parboot(fm.oven.1, statistic = SSE, nsim = 20)

t0 = 72.24217
76.2, 97.5
65.1, 89.7
49.9, 77.3
56.6, 60
88, 76.9
78.3, 83.5
79.5, 73.4
76.4, 85.1
77.4, 68.3
85.6, 76

> pb

Call: parboot(object = fm.oven.1, statistic = SSE, nsim = 20)

Parametric Bootstrap Statistics:
      t0 mean(t0 - t_B) StdDev(t0 - t_B) Pr(t_B > t0)
SSE 72.2          -3.80             11.6        0.714

t_B quantiles:
    0% 2.5% 25% 50% 75% 97.5% 100%
SSE 50   53  72  77  84    94   97

t0 = Original statistic compuated from data
t_B = Vector of bootstrap samples

> plot(pb, main = "Goodness of fit Parametric Bootstrap")
```

The above call to `plot` with a parametric bootstrap object as the argument produces a useful graphic for assessing goodness of fit (see Figure 2). The plot suggests that the model can sufficiently explain these data.

4. Future directions for unmarked development

unmarked has become a stable and useful platform for the analysis of ecological data, but several areas of development could improve its utility. Bayesian estimation could be implemented for many of these models. This would provide a useful comparison with the maximum likelihood results. Random effects are an often requested feature from users that will likely be incorporated into a future version. Random effects are useful for modeling heterogeneity

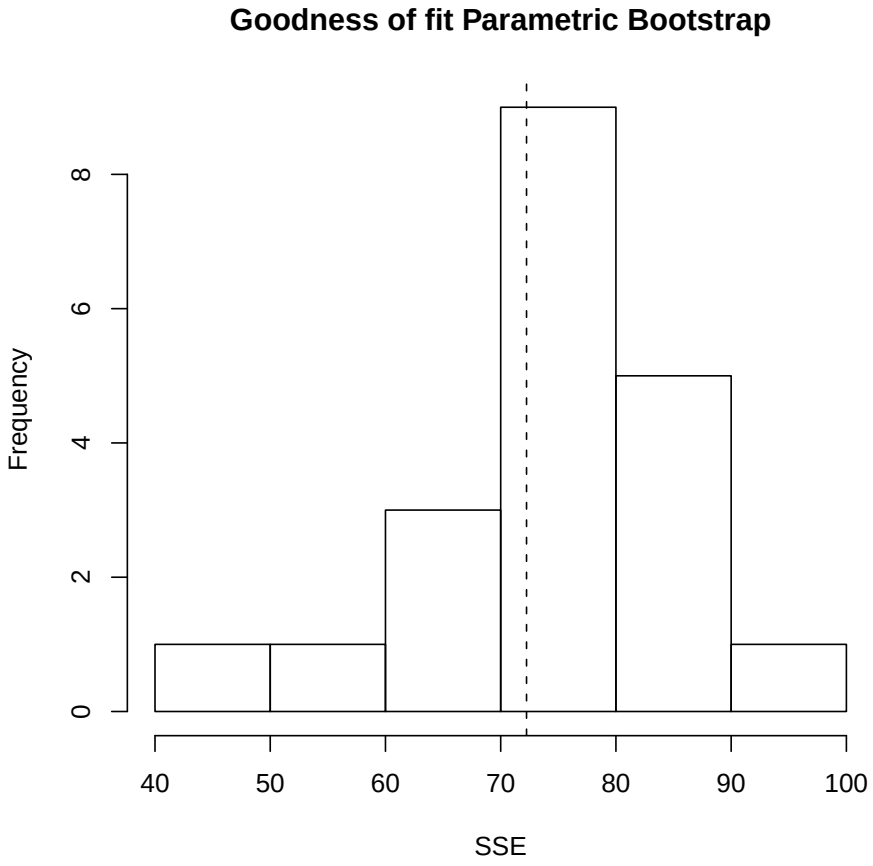

Figure 2: Graphically assess model fit by parametric bootstrapping the sum of squared residuals. The dashed line is the observed sum of squared residuals.

among sites that is not accounted for by covariates alone. For maximum likelihood methods, the Laplace approximation to the integrated likelihood approach is likely to be fruitful and the incorporation of random effects into Bayesian estimation is even more straightforward. Also, **unmarked** can easily be extended to add new TLEMs as they are published and found to be useful.

Acknowledgements

The authors thank Andy Royle of the United States Geological Survey's Patuxent Wildlife Research Center, who provided initial funding and inspiration for this work.

References

Affiliation:

Ian Fiske
Department of Statistics
North Carolina State University
2311 Stinson Drive
Campus Box 8203
Raleigh, NC 27695-8203
E-mail: ijfiske@ncsu.edu

Richard Chandler
Department of Natural Resources Conservation
University of Massachusetts
Room 225
160 Holdsworth Way
Amherst, MA 01003-9285
E-mail: rchandler@nrc.umass.edu

Journal of Statistical Software http://www.jstatsoft.org/
published by the American Statistical Association http://www.amstat.org/
Volume VV, Issue II *Submitted:* yyyy-mm-dd
MMMMMM YYYY *Accepted:* yyyy-mm-dd