Fast and flexible Bayesian Jolly Seber models and application to populations with transients

James F. Saracco[1,*] and Charles B. Yackulic[2]

[1]The Institute for Bird Populations, Petaluma, California 94953 USA

[2]U. S. Geological Survey, Southwest Biological Science Center, Flagstaff, Arizona 86001 USA

*Email: jsaracco@birdpop.org

Abstract

1. Jolly Seber (JS) models are an appealing class of capture-recapture models for modeling open populations because they allow for inferences about an array of population parameters, including abundance, survival, and recruitment. Multiple formulations of JS models have been developed and include both maximum likelihood and Bayesian approaches. Bayesian approaches offer greater flexibility; however, they are often extremely slow when applied to moderate to large populations because the latent states of all observed individuals, as well as any potential unobserved individuals (parameter-expanded data augmentation), must be simulated at each time step.
2. JS model implementations can be sped up dramatically through marginalization, whereby the conditional likelihoods of states are tracked for unique capture histories, rather than the simulated states of all individuals. Here we describe a marginalized implementation of a multistate JS model and compare its performance to that of an implementation using simulated discrete states. We fit models to data generated under two scenarios, one in which no data was missing and another in which 25% of data

was randomly removed. To illustrate how marginalization can accommodate more complex models and datasets, we describe a modified version of the model for application to populations with transients and fit the model to simulated data and to data collected on Kentucky Warblers (*Geothlypis formosa*) as part of the Monitoring Avian Productivity and Survivorship (MAPS) program.

3. Both the discrete latent state and marginalized JS model implementations performed similarly with respect to bias and coverage; however, the marginalized implementation was roughly 1,000 times faster when no data were missing and 100 times faster when 25% of data was missing. The marginalized model accommodating transients also converged quickly and a more complex version of the model applied to the larger multi-site Kentucky Warbler data set yielded useful estimates of most parameters within a reasonable time frame (hours).
4. Gains in efficiency provided by marginalization shown here should stimulate additional study and development of this useful class of models and provide new opportunities for real-world applications to large complex data sets.

keywords: capture-recapture; discrete latent state, Hidden Markov model, marginalization, Monitoring Avian Productivity and Survivorship, open population, recruitment, residency, survival, transient model.

Introduction

Open-population capture-recapture models are important tools for estimating demographic parameters of wildlife populations (Nichols 2005). These models include two basic classes: (1) Cormack-Jolly-Seber (CJS) models, which condition on first capture and provide estimates of apparent survival probability (Cormack 1964); and (2) Jolly-Seber (JS)

models (Jolly 1965, Seber 1965), which are informed by entire capture histories (i.e., including occasions both prior to and after first capture). JS models are appealing because they make more complete use of available data and can provide inferences on an expanded set of demographic parameters including abundance and recruitment-related parameters. JS models assume that marked and unmarked animals have equal capture probability, which may limit their applicability in some situations (e.g., mark-resight studies). However, this assumption is likely reasonable for a broad array of studies that implement the same sampling protocol for both captures and recaptures (e.g., Robinson et al. 2009).

A variety of formulations of both CJS and JS models have been developed and include both maximum likelihood and Bayesian approaches (Pradel 1996, Schwarz and Arnason 1996, Link and Barker 2005, Dupuis and Schwarz 2007, Schofield and Barker 2008, Royle and Dorazio 2012, Tenan et al. 2014, Hostetter et al. 2021). Bayesian implementations offer several advantages over maximum likelihood methods, including ability to incorporate prior information, flexibility for handling hierarchical random effects, and relative ease with which derived quantities and their uncertainties can be calculated. A common Bayesian approach, known as state-space modeling or discrete latent state modeling (Royle 2008, Yackulic et al. 2020), involves simulating the state of individuals (e.g., alive/dead) and the conditional probability of their detection based on Bernoulli or categorical models applied to a set of observed capture histories. For JS models, the data set must be augmented to include a set of capture histories with no detections that represent individuals never observed to allow for modeling of both the survival and entry processes (Kéry and Schaub 2011). A drawback of this approach, however, is that computation times using Markov Chain Monte Carlo (MCMC) techniques can be prohibitive for many applications, particularly when the augmented data set includes many individuals or levels of stratification (e.g., sites or spatial strata).

Bayesian open-population models with discrete latent states can be sped up dramatically through marginalization (Yackulic et al. 2020). Rather than tracking the state of all individuals at each time step, these marginalized models only require tracking likelihoods of possible states given the demographic parameters for a reduced data set of unique capture histories and their frequencies. Yackulic et al. (2020) provide a thorough introduction to the marginalization in Bayesian population models, as well as code and examples for CJS models. Here we extend this work in the context of JS models. We focus on the multistate expression of the JS model described in Royle and Dorazio (2012), which is readily adaptable to a variety of model extensions involving additional states (Kéry and Schaub 2011). We apply the model to simulated complete data sets and data sets with random missing data, a common scenario in multi-site multi-collaborator data sets (e.g., Saracco et al. 2010), and compare parameter estimates and run times between the marginalized models and discrete latent state implementations using the popular open-source software program JAGS (Plummer 2003).

We also describe an extension of the basic marginalized multistate JS model to accommodate situations where not all sampled individuals are part of the resident populations. This model variant can both improve inferences about survival, recruitment, and population size, as well as allow estimation of residency probability and proportions of residents in populations (or, conversely, transience probability and proportions of transients), which may also often be of ecological interest (Wilson et al. 2018, Genovart and Pradel 2019, Oro and Doak 2020). Accounting for negative bias in survival estimates due to transiency, and estimation of residency or transiency probabilities, has been treated within the context of CJS models (Pradel et al. 1997, Nott and DeSante 2002, Hines et al. 2003, Royle and Dorazio 2008, Saracco et al. 2010). However, transiency or residency probability estimates from these models only apply to newly marked individuals and so may not be representative of the larger

population because they depend, to some extent, on the proportions of previously marked individuals still alive and available for capture (Hines et al. 2003). Estimation of parameters related to the dynamics of resident and transient individuals within seasons has been developed within an 'open robust design' framework (Ruiz-Gutierrez et al. 2016); however, these models require additional data and model complication that may be unnecessary if primary interest is on between season dynamics. Treatment of these parameters in the JS context represents an important advance because it allows for disentangling of season-specific observation (first capture) and state (residency) processes within a simpler multi-season open population model.

We apply this 'transient JS' model to simulated data as well as to regional data from a migratory bird species, Kentucky Warbler (*Geothlypis formosa*), from bird-banding stations operated as part of the Monitoring Avian Productivity and Survivorship (MAPS) program (DeSante et al. 2004) in the Central Hardwoods Bird Conservation Region of the United States (Sauer et al. 2003).

Materials and Methods

Multistate Jolly Seber models: structure and data

Both the discrete latent state and multistate JS model assume a "superpopulation" (*sensu* Schwarz and Arnason 1996), N_{super} , representing the number of individuals ever alive in the population. Membership in N_{super} is determined as a function of parameters describing the entry process and an augmented data set consisting of *M* individuals, including all those observed during the study, as well as any unobserved individuals that could have been members of the superpopulation. While the number of unobserved individuals to include in the augmented data set is somewhat arbitrary, the choice is not innocuous, particularly when

using a discrete latent state implementation – too many and run times will be unnecessarily increased, too few and parameter estimates may be biased.

Following Kéry and Schaub (2011) and Royle and Dorazio (2012), we model the entry process in terms of a removal entry probability, γ, which represents the probability of entry for a member of M conditional on not having previously entered (see Appendix S1 for an alternative parameterization). Once entered, the probability of remaining alive and in the population is determined by a survival parameter, ϕ. We can represent the latent state of the $i = 1, \ldots, M$ individuals across $t = 0, \ldots, T$ time periods with a matrix, $\mathbf{S}$, with entries, $s_{i,t}$ that can take values of: 1 = not yet entered into the population, 2 = alive and in the population, and 3 = dead. The number of time periods, T, includes all observed sampling occasions as well as a dummy occasion at $t = 0$ to allow modeling of initial entry into the population. All $s_{i,0}$ are set equal to 1 to indicate that none have entered prior to the first observed time period at $t = 1$. The hidden Markov process determining $s_{i,t}$ can be represented with an array, $\mathbf{\Omega}$, with individual entries $\omega_{j,k,t}$ representing probabilities of transitioning from latent state j (rows) in time t to latent state k (columns) in time $t + 1$:

$$\omega_{j,k,t} = \begin{pmatrix} 1-\gamma_t & \gamma_t & 0 \\ 0 & \phi_t & 1-\phi_t \\ 0 & 0 & 1 \end{pmatrix}.$$

The data informing the latent states consist of a set of observed and unobserved (i.e., augmented) capture histories and, for the discrete latent state model, can be represented with a $M \times T$ matrix, $\mathbf{Y}$, with entries $y_{i,t}$. For the marginalized model, the $y_{i,t}$ are summarized to a more compact matrix, $\mathbf{YS}$, with entries $ys_{m,t}$ that represent $m = 1, \ldots, MS$ unique capture histories over the T years. The data informing the marginalized model are then the $ys_{m,t}$ and their observed frequencies. The entries of $\mathbf{Y}$ and $\mathbf{YS}$ can take values of 1 (captured) or 2s (not captured). For computational purposes, we set all $y_{i,0} = 2$ and $ys_{m,0} = 2$ for the initial dummy capture occasion. These capture histories are linked to latent states by a second array,

P, with entries $\rho_{j,o,t}$ indexed by latent state *j* (rows), observed state *o* (columns), and period of observation *t* (not counting the dummy capture occasion), with time-specific capture probabilities represented by p_t:

$$\rho_{j,o,t} = \begin{pmatrix} 0 & 1 \\ p_t & 1 - p_t \\ 0 & 1 \end{pmatrix}.$$

The principal difference between the discrete latent state and marginalized JS model implementations is the definition of the likelihoods. In the discrete latent state model, the true latent state is simulated for each individual and occasion for all $t > 0$ based on a categorical model, $s_{i,t} \sim Cat\left(\omega_{z[i,t-1],1:3,t-1}\right)$. The likelihood is then determined based on a categorical model of the observations conditional on the latent state, $y_{i,t} \sim Cat\left(\rho_{s[i,t],1:2,t}\right)$.

The likelihood for the marginalized JS model is defined in terms of an array, **Z**, which tracks capture history likelihoods conditional on the demographic parameters. Its dimensions are defined (in order) by unique capture histories, $t = 0, \ldots, T$ sampling occasions (again, including the dummy occasion at $t = 0$), and $s = 1{:}\,3$ states. Each entry, $\zeta_{m,t,s}$ is the likelihood that an individual with capture history *m* was in state *s* at time *t* conditional on the observed capture history through time *t* and values of the process and observation parameters. As in the discrete latent state model, no individuals have entered the population at $t = 0$. Thus, we set $\zeta_{m,0,1} = 1$ and $\zeta_{m,0,2:3} = 0$ so that all individuals are available prior to the study. For each subsequent time step, $\zeta_{m,t,s}$ is determined by first taking the inner product of $\zeta_{m,t-1,1:3}$ and the column vector taken from s^{th} column of the state transition matrix and then multiplying this quantity by the probability of the observed state given the state *s* (i.e., $\zeta_{m,t,s} = \langle \zeta_{m,t-1,1:3}\,,\ \omega_{1:3,s,t-1} \rangle \times \rho_{s,o[m,t],t}$. For example, at the first true capture period (*t* = 1), the observation probability for capture histories with capture at that time period (i.e., $y_{m,1} = 1$) would be calculated as $\zeta_{m,1,1} = \zeta_{m,0,1}\omega_{1,1,0}\rho_{1,1,1} = 1 \times (1 - \gamma_0) \times 0 = 0$ for individuals that did not enter (i.e., $s = 1$); $\zeta_{m,1,2} = \zeta_{m,0,1}\omega_{1,2,0}\rho_{2,1,1} = 1 \times \gamma_0 \times p_1 = \gamma_0 p_1$

for individuals that entered (i.e., $s = 2$); and $\zeta_{m,1,3} = \zeta_{m,0,1}\omega_{1,3,0}\rho_{2,1,1} = 1 \times 0 \times 0 = 0$ for dead individuals (i.e., s = 3). Thus, the only possible transition for individuals captured at the first true capture occasion is entry into the population.

The likelihood of each capture history is determined by summing the likelihoods of all possible states, s, after the final capture occasion, $\sum_{s=1}^{3} \zeta_{\mathrm{m},T,s}$, weighted by the frequency of the mth capture history. In BUGS language, as implemented in popular software programs such as WinBUGS (Lunn et al. 2000), OpenBUGS (Lunn et al. 2009), JAGS (Plummer 2003), and NIMBLE (de Valpine et al. 2017), a modified version of the "ones trick" can be used to update the posterior with these values. See Yackulic et al. (2020) and Appendix S1 for detail; Yackulic et al. (2020) also provide the appropriate approach to updating the posterior using the program Stan (Carpenter et al. 2017).

For both the discrete latent state and marginalized models, unconditional time-specific entry probabilities from the superpopulation to the population, b_t are derived as a function of the entry probabilities conditional on not having previously entered, c_t, where $c_0 = \gamma_0$ and $c_t = \gamma_t \prod_{j=0}^{t-1}(1 - \gamma_j)$ for $t \geq 1$, and the inclusion probability (i.e., probability of being a member of the superpopulation), ψ (Kéry and Schaub 2011): $b_t = c_t/\psi$. In discrete latent state models, secondary parameters (i.e., derived quantities) can be calculated directly from the modeled latent states. For example, time-specific numbers of newly entered individuals (i.e., newborns or immigrants), B_t, can be derived by counting the $z_{i,t} = 2 | z_{i,t-1} = 1$, and total time-specific abundance, N_t, can be calculated by counting the $z_{i,t} = 2$ for each sample of the posterior distribution.

Alternatively, secondary parameters can be derived as functions of primary (i.e., modeled) parameters, other secondary parameters, and/or the input data; and this approach can be applied to either model type. For example, numbers of new entrants can be estimated as $B_t = c_t M$, and the superpopulation size, N_{super}, can then be derived as $\sum_{t=0}^{T-1} B_t$ (Schwarz

and Arnason 1996). Time-specific abundance can be estimated as $N_1 = B_0$ for the initial sampling period and $N_t = B_{t-1} + \phi_t N_{t-1}$ for $t > 1$, or alternatively could be estimated with the canonical estimator, $N_t = n_t/p_t$, where the n_t represents a count of the number of individuals observed alive at time *t* (i.e., summing the frequencies of all $y_{i,t} = 1$ or $ys_{m,t} = 1$). Other quantities of interest that can be derived include population growth rate, $\lambda_t = N_{t+1}/N_t$, and per capita entry rate, $f_t = B_t/N_t$ (or, alternatively, $f_t = \lambda_t - \phi_t$; Pradel 1996, Link and Barker 2005). Additional methods for deriving secondary parameters from marginalized models are also possible. For example, annual abundance can be derived by simulating latent states from reconstituted individual capture histories (i.e., converting $\mathbf{YS}$ back to $\mathbf{Y}$) and summing them in a manner analogous to what is described above for the discrete latent state models (see Appendix S2 for detail).

Marginalized multistate Jolly Seber model with transients

Capture-recapture studies often include individuals that are sampled but are not resident in the local population. These transient individuals have been defined in the context of CJS models as individuals having zero probability of being alive and available for capture following initial capture (Pradel et al. 1997). In the JS context, however, transients can be defined more generally as individuals appearing only once and never returning to the sampling area. Thus, interpretation of transience and residency probabilities in the JS context is more ecologically relevant because it refers to the larger population, not just the population of newly marked individuals. We can accommodate transiency in the multistate JS model by adding a state that allows for entry as either a resident or a transient, such that there are now four possible states, including: 1 = not yet entered; 2 = alive resident; 3 = alive transient; and 4 = dead. Distinguishing between the two alive states requires an extra parameter π, denoting the probability of being a resident. Thus, individual entries in the transition and capture probability arrays become:

$$\omega_{j,k,t} = \begin{pmatrix} 1-\gamma_t & \gamma_t\pi & \gamma_t(1-\pi) & 0 \\ 0 & \phi_t & 0 & 1-\phi_t \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \end{pmatrix} \text{ and } \rho_{j,o,t} = \begin{pmatrix} 0 & 1 \\ p_t & 1-p_t \\ p_t & 1-p_t \\ 0 & 1 \end{pmatrix}.$$

The residency probability parameter, π, is estimated in the marginalized multistate transient JS model based on $N_i^R \sim Bin(\pi, FR_i)$, where N_i^R is the number of residents represented by unique capture history i. In some cases, the residency state can be partially observed. For example, within a 'robust design' sampling framework (Pollock et al. 1990), whereby multiple secondary sampling occasions are completed within each primary sampling occasion, individuals observed on multiple secondary sampling occasions during the primary sampling occasion in which they were marked can be used as an indicator of residency (Nott and DeSante 2002, Hines et al. 2003). To accommodate this scenario, and following logic developed for the discrete latent state version of a transient CJS model (Saracco et al. 2010), we can define an observation model for residency as: $r_i \sim Bin(\kappa, N_i^R)$, where r_i is the number of observed residents for capture history i, κ is the probability of observing residency, and N_i^R is the true number of residents for capture history i. Note that in contrast to the CJS scenario, where residency observation and state probabilities are specific to years of first capture and allow for temporal modeling, here both κ and π are time-constant. Nevertheless, we can derive numbers of transients and residents entering the population at each time step as $B_t^{tr} = c_t * (1-\pi) * M$ and $B_t^{res} = c_t * \pi * M$, respectively. For transients, the number of entries is equivalent to the number of transients in the population at any time step; i.e., $B_t^{tr} = N_t^{tr}$. At $t = 1$, the total number of residents is also the same as the number of resident entries, $N_1^R = B_1^R$; the number of residents can be derived as $N_t^R = B_t^R + \phi_t * N_{t-1}^R$ for subsequent time steps. The proportion of residents at any given time can then be derived as $prop(res)_t = N_t^R/(N_t^R + N_t^{tr})$.

A critical assumption of this formulation of the transient JS model is that p_t, the probability of capture at least once across the season, is the same for both transient and resident individuals. This is plausible if most individuals enter the population early in the season and we think of p_t as an initial capture probability, π as the probability of persisting for multiple sampling periods, and κ as the probability of capturing an individual that persists for multiple periods. Nevertheless, we acknowledge that in many cases residents and transients will have differential exposure to capture, resulting in violations of the assumption of equal p_t between these two groups. Accounting for these differences can be accommodated with additional model development and data from secondary samples within seasons (e.g., Ruiz-Gutierrez et al. 2016). For simplicity, we do not consider this additional complication here.

Discrete v. marginalized multistate JS models: a simulation study

To compare the relative speed and efficacy of discrete latent state and marginalized JS models, we simulated 100 capture-recapture data sets. As in Kéry and Schaub (2011), we assumed a superpopulation size of $N_{super} = 400$ across $T - 1 = 7$ capture occasions, a constant survival probability of $\phi = 0.70$, entry probabilities of $b_1 = 0.34$ and $b_{2...7} = 0.11$, and a constant capture probability of $p = 0.50$. We assumed $U(0,1)$ prior distributions for survival, ϕ, capture probability p, and entry probabilities b_t. Data sets were augmented with 500 all-ones capture histories to account for unobserved individuals for a total of M capture histories (observed + unobserved). These individual capture histories made up the basic data sets for the discrete latent state JS model, an $M \times T$ matrix, where T represents the number of sampling occasions, including the dummy occasion. For the marginalized model, data sets (including the augmented individuals) were aggregated to MS unique observed capture histories and their frequencies.

To explore potential effects of missing data, we randomly assigned 25% of the individual × capture occasion data generated in the simulated data sets to be missing. For the marginalized models, capture histories were then collapsed to unique capture history × effort (missing or not). Missing data in the two model types are handled differently. For the discrete latent state models, missing data values are imputed with the categorical models of the state and observation process in the same way they are for the complete data situations, which should result in relatively little increases in model run times. However, for the marginalized models, where state and capture probabilities are tracked with deterministic nodes, missing data can be accommodated by setting capture probabilities to zero for missed occasions. This requires tracking capture probabilities and conditional state likelihoods across an additional dimension allowing for different capture probabilities among individuals sampled at different times, which may result in relatively greater loss in efficiency. An alternative approach, which we do not explore here but that could also have implications for model run times, would be to define a multistate model for p_t, whereby p_t can take 'available' or 'unavailable' states depending on whether sampling occurred (e.g., Lovich et al. 2014).

For the discrete latent state models, we derived secondary parameters as functions of the summed latent states as in Kéry and Schaub (2011). For the marginalized models, we derived secondary parameters as analytical functions of the primary parameters and data or other second-order states, as described above, as a simulation study based on the primary parameter values used here showed better credible interval coverage, faster implementation time, and less biased parameter estimates compared to alternative methods of deriving second-order states and secondary parameters (Appendix S2).

To assess models and data sets, we compared the ability of models to recover data generating values, 95% credible interval coverage, credible interval widths, and model run times. We implemented all models in JAGS (Plummer 2003) with the jagsUI package

(Kellner 2015) in R (R Core Team 2021). R code for simulating data and implementing models is provided in the data package (Saracco and Yackulic 2023), and a detailed description of model code is provided in Appendix S1.

Transient marginalized multistate JS models

We applied a transient marginalized multistate JS model to 100 simulated data sets using the same parameters as the marginalized JS model above, but with the added residency parameters set to $\pi = 0.6$ and $\kappa = 0.4$. Simulation and model details are provided in the data package.

We applied a version of this model to capture-recapture data from 2912 adult Kentucky Warblers (*Geothlypis formosa*) banded between 1993 and 2015 in the Central Hardwoods Bird Conservation Region in the United States (Sauer et al. 2003) as part of the Monitoring Avian Productivity and Survivorship (MAPS) program. These data were collected at 51 MAPS stations, which on average operated ~ 10 yrs (sd = 6) of the 23-yr time span, resulting in ~ 42% (28058 of 66976) of the individual × year data points missing. For the analysis, these data were summarized to yield 876 unique capture histories. We used a $Beta\big(1/(T-1), 2-t/(T-1)\big)$ prior distribution on γ_t due to the missing data and large number of sampling periods (Dorazio 2020). Initial attempts using $U(0,1)$ priors for γ_t with MAPS data resulted in marginal posterior distributions on N peaking at M, regardless of how many unobserved individuals the data set was augmented with. To allow for spatial variation in capture probability, we modeled $p_{sta[i]}$ with a logit-linear model with intercept and a random station effect distributed as $Norm(0, \sigma^2)$. In addition, we included annual variation in survival by modeling ϕ_t with a logit-linear model with intercept and a random year effect distributed as $Norm(0, \sigma^2)$. We applied standard vague priors for intercepts ($Norm(0,0.01)$) and variances of random effects ($Unif(0,10)$).

Results

Discrete v. marginalized multistate JS models

Median estimates of the data generating parameter values were successfully recovered for both the discrete latent state and marginalized multistate JS model implementations (Fig. 1). Credible interval coverage was close to nominal at the 95% level for most parameters and scenarios, although it was slightly lower for the N_{super} and p parameters (Table 1; Fig. 1). Coverage for the N_{super} and p parameters for models with missing data was especially low (~ 80-90%). The abundance estimates, N_t and N_{super}, tended to be biased high for all models and data scenarios. Median bias of posterior medians for the abundance parameters was lowest for the discrete latent state model with complete data (+ 3-7%), slightly higher for the marginalized model with complete data (+ 5-8%), and greater still for the models with missing data (+ 6-12% for the discrete latent state model and + 7-14% for the marginalized model). Estimates of capture probability, *p*, tended to be biased low and was similar between models (median – 5% bias in posterior *p* medians for both models with complete data and – 6% for models with missing data). Credible interval widths were essentially identical between the discrete and marginalized models for the entry, survival, and capture probability parameters; however, widths were ~ 10-20% greater for derived parameters, such as N_{super} (Table 1). The marginalized models ran much faster than the discrete latent state models, reaching 1000 effective samples for the least converged parameter ~ 100 × (missing data model) to 1000 × faster than the discrete latent state versions (Fig. 2).

Transient model applied to simulated data

The model converged quickly, with a median time to achieve 1000 effective samples of the least converged parameter of 0.5 minutes. Parameter estimates were very similar to the data generating values, suggesting that the model can provide reliable inferences about residency probability, numbers of residents in the population, survival and recapture

probabilities under the assumption of equal p_t between residents and transients. As for the basic model version, however, estimates of capture probability tended to be biased low and population size high (Fig. 3). Residency probability also tended to be biased high (Fig. 3a). Coverage of 95% credible intervals was relatively low for capture probability and superpopulation size of residents (between 80% and 85%, Fig. 3c).

Application to MAPS data

Based on a model run with 3 MCMC chains of 40000, burn-in of 10000, and thinning rate of 3 (for a total of 30000 saved posterior samples), the model took approximately 24 hours to run, considerably longer than our simpler simulation examples. Most parameters converged relatively quickly with the median time to achieve 1000 effective samples of 4 hours; however, a few parameters took longer with the least converged parameter taking 3.7 days to achieve 1000 effective samples. All parameters had $\hat{R} < 1.01$, suggesting satisfactory convergence.

Model results suggested that survival probability was relatively consistent across the study (ϕ=0.5 [0.46, 0.54]; σ=0.09 [0, 0.3]; Fig. 4a). The derived recruitment rate was much more variable, particularly for the first and last time intervals (Fig. 4b). However, precision of the recruitment estimates was also lower for those intervals and likely reflect the low number of stations operated during the first and last years of the time series (8 and 3, respectively). Annual population change and abundance matched patterns of recruitment (Figs. 4c, d), suggesting the importance of this demographic rate in driving population dynamics. Residency probability, π, was estimated to be (π = 0.53 [0.44, 0.64], and the proportion of residents in the population was relatively consistent across years, again with the exception of first and last years (Fig. 4e).

Discussion

Bayesian modeling with MCMC estimation provides a flexible framework for implementing capture-recapture models with latent states and complex hierarchical model structures. However, because common Bayesian implementations of these models require simulating discrete latent states of each individual at each time step, they can be slow to converge and require long run times for moderate to large data sets (Yackulic et al. 2020). This is especially true for JS models, where observed capture histories must be augmented with potentially large numbers of unobserved individuals to encompass the population of interest. Here we show that through marginalization, whereby the individual capture histories are aggregated to unique capture histories (or unique capture history × covariate) and their frequencies, and state probabilities, rather than individual states, are tracked, parameter estimates can be recovered in a fraction of the time. This gain in efficiency greatly expands the potential applications for this useful class of capture-recapture model.

Because JS models do not condition on first capture, they require an assumption of equal capture probability for captured and uncaptured individuals, which may limit their utility for certain sampling scenarios (e.g., mark-resighting studies). However, whenever this assumption is reasonable, these models provide an attractive alternative to CJS models, because they enable inferences about an expanded suite of demographic parameters including recruitment, population size, and population dynamics. As proposed in the original formulations of Jolly (1965) and Seber (1965), population size can be estimated within a CJS modeling context by simply dividing the number of individuals captured by the recapture probability estimate (Dugger et al. 2004, McDonald et al. 2005 p. 241, see Yackulic et al. 2014, Appendix C for a complex application). However, this approach can lead to population size estimates that suggest negative recruitment because entry parameters are not included in

the likelihood (Schwarz and Arnason 1996). By modeling both the recruitment and survival processes, JS approaches provide a means of preventing such inconsistencies. Other options that do condition on capture, yet are still able to provide inferences on recruitment through modeling capture histories both forward and backward in time, offer an alternative approach (Pradel 1996, Link and Barker 2005). These 'reverse-symmetry' models offer an advantage over JS models in that the recruitment or population growth parameters can be included in the likelihood and so modeled directly, rather than derived from the survival and population size parameters (Tenan et al. 2014, Saracco et al. 2020). In addition, because they are based on applying time-specific multinomial likelihoods to a compact set of sufficient statistics, they too can be implemented quickly using Bayesian methods. Nevertheless, these reverse-symmetry models do not allow the same level of flexibility as the multi-state JS models for handling complexities such as transiency or demographic rates that depend on states that change over time [e.g., age class or breeding status; Kendall et al. (2019), Hostetter et al. (2021)]. Finally, while these models can provide inference about population dynamics parameters, they are largely unable to inform population size (but see Tenan et al. 2019).

Median parameter estimates from our simulation study were very similar between the discrete latent state and marginalized JS models for the set of parameter values we considered in our simulations. However, both model implementations showed slight negative bias in capture probability estimates and positive bias in population size estimates. These biases were magnified for model runs with random missing data. We suggest that additional study is needed to better understand the potential effects of missing data, including different amounts and types of missing data (e.g., random v. consecutive missed years at particular sites), on inferences across a range of combinations of parameter values. Understanding the consequences of missing data is especially important for monitoring programs, such as

MAPS, where as much as half the sampling occasions can commonly be missed due to stations entering and leaving the program over the years.

Credible intervals were wider for the marginalized v. discrete latent state models for the population size and other parameters derived as deterministic functions of population size. Nevertheless, these estimates exhibited better coverage compared to the discrete latent state model or marginalized model with parameters derived from conditional estimates applied to reconstituted capture histories (Appendix S2). Given the greater speed of implementation, better coverage, and lack of bias in year-specific population size, recruitment, and population change estimates, we suggest that this approach appears to be the most generally useful among those considered here for implementing models and estimating derived parameters.

We described a variation of the basic marginalized multistate JS model to illustrate the versatility of the basic modeling framework to address a common complication of many capture-mark-recapture programs that involves the sampling of individuals that are not part of the target population. This 'transient' version of the marginalized multistate JS model assumes equal capture probability between residents and transients, which may be unrealistic in many situations; however, because capture probability is informed here by between-year recaptures, we suggest that it is unlikely to bias estimates of resident population size. Higher resident than transient recapture probabilities would nonetheless be expected to result in underestimates of transient populations and overestimates of the proportion of residents in populations. Additional within-season data and model development (e.g., Ruiz Gutierrez et al. 2016) will be required to provide a more complete understanding of resident and transient components of populations.

The version of the transient JS model that we describe here successfully recovered parameter values from our simulations, although the same biases in capture probability and superpopulation estimates seen in the basic JS version were also evident in the transient

model, and there was some evidence in bias in the residency probability parameter as well. We suggest that additional study is needed to assess how biases may change across different combinations of parameter values.

While our application to the MAPS Kentucky Warbler data set was much slower than the simple simulation scenarios we considered, this was not entirely unexpected given the larger dimensions of the data set (~ 20 x the number of rows representing unique site x capture histories and > 3 x the number of sampling occasions compared to the simulated transient model data sets), random effects structure for the survival and capture probability models, and large number of missed sampling occasions. Even with the longer run time, reasonable inferences for most parameters were obtained within a few hours, demonstrating the potential utility of this modeling framework for more complicated real-world examples.

Ideally, study design, objectives, and characteristics of the data set at hand should dictate possible model applications without having to consider constraints of long implementation times. We hope that the marginalized versions of the JS model described here will stimulate additional study and development of this useful class of models and open new potential applications to large complex data sets.

Acknowledgements

Funding was provided by the Knobloch Family Foundation. R. Siegel and S. Albert provided logistical support. We are grateful to the many MAPS cooperators and to D. DeSante for his work in developing and managing the MAPS program. D. Kaschube helped to prepare the MAPS data set with support from E. Cox, L. Helton, and R. Taylor. R. B. Taylor and W. Kendall provided helpful comments that improved the manuscript. Any use of trade, product, or firm names in this publication is for descriptive purposes only and does not imply endorsement by the U.S. Government. This is Contribution Number 731 of The Institute for Bird Populations.

Conflict of Interest Statement

The authors declare no conflict of interest.

Author Contributions

J. Saracco and C. Yackulic conceived the ideas and designed methodology; J. Saracco prepared and analysed the data and led the writing of the manuscript. Both authors contributed critically to the drafts and gave final approval for publication.

Data Availability

R code and data needed to reproduce results are archived in the Open Science Framework https://doi.org/10.17605/OSF.IO/Q3458 (Saracco and Yackulic 2023).

Literature cited

Carpenter, B., A. Gelman, M. D. Hoffman, D. Lee, B. Goodrich, M. Betancourt, M. Brubaker, J. Guo, P. Li, and A. Riddell. 2017. Stan: A probabilistic programming language. Journal of Statistical Software 76:1–32.

Cormack, R. M. 1964. Estimates of survival from the sighting of marked animals. Biometrika 51:429–438.

de Valpine, P., D. Turek, C. Paciorek, C. Anderson-Bergman, D. Temple Lang, and R. Bodik. 2017. Programming with models: Writing statistical algorithms for general model structures with NIMBLE. Journal of Computational and Graphical Statistics 26:403–413.

DeSante, D. F., J. F. Saracco, D. R. O'Grady, K. M. Burton, and B. L. Walker. 2004. Methodological considerations of the monitoring avian productivity and survivorship (MAPS) program. Studies in Avian Biology 29:28–45.

Dorazio, R. M. 2020. Objective prior distributions for jolly-seber models of zero-augmented data. Biometrics 76:1285–1296.

Dugger, K. M., J. Faaborg, W. J. Arendt, and K. A. Hobson. 2004. Understanding Survival and Abundance of Overwintering Warblers: Does Rainfall Matter? The Condor 106:744–760.

Dupuis, J. A., and C. J. Schwarz. 2007. A bayesian approach to the multistate jolly–seber capture–recapture model. Biometrics 63:1015–1022.

Genovart, M., and R. Pradel. 2019. Transience effect in capture-recapture studies: The importance of its biological meaning. PLOS ONE 14:e0222241.

Hines, J. E., W. L. Kendall, and J. D. Nichols. 2003. On the use of the robust design with transient capture-recapture models. The Auk 120:1151.

Hostetter, N. J., N. J. Lunn, E. S. Richardson, E. V. Regehr, and S. J. Converse. 2021. Age-structured jolly-seber model expands inference and improves parameter estimation from capture-recapture data. PLOS ONE 16:1–19.

Jolly, G. M. 1965. Explicit estimates from capture-recapture data with both death and immigration-stochastic model. Biometrika 52:225–247.

Kellner, K. 2015. jagsUI: A Wrapper Around ’rjags’ to Streamline ’JAGS’ Analyses.

Kendall, W. L., S. Stapleton, G. C. White, J. I. Richardson, K. N. Pearson, and P. Mason. 2019. A multistate open robust design: Population dynamics, reproductive effort, and phenology of sea turtles from tagging data. Ecological Monographs 89:e01329.

Kéry, M., and M. Schaub. 2011. Bayesian Population Analysis using WinBUGS. Page 554. 1st Edition.

Link, W. A., and R. J. Barker. 2005. Modeling association among demographic parameters in analysis of open population capture-recapture data. Biometrics 61:46–54.

Lovich, J. E., C. B. Yackulic, J. Freilich, M. Agha, M. Austin, K. P. Meyer, T. R. Arundel, J. Hansen, M. S. Vamstad, and S. A. Root. 2014. Climatic variation and tortoise survival: Has a desert species met its match? Biological Conservation 169:214–224.

Lunn, D. J., A. Thomas, N. Best, and D. Spiegelhalter. 2000. WinBUGS - A Bayesian modelling framework: Concepts, structure, and extensibility. Statistics and Computing 10:325–337.

Lunn, D., D. Spiegelhalter, A. Thomas, and N. Best. 2009. The BUGS project: Evolution, critique and future directions. Statistics in Medicine 28:3049–3067.

McDonald, T. L., S. C. Amstrup, E. V. Regehr, and B. F. Manly. 2005. Examples. *in* S. C. Amstrup, T. L. McDonald, and B. F. Manly, editors. Handbook of capture-recapture analysis. Princeton University Press, Princeton, New Jersey.

Nichols, J. D. 2005. Modern open-population capture-recapture models. Pages 88–123 *in* S. C. Amstrup, T. L. McDonald, and B. F. J. Manly, editors. Handbook of capture-recapture analysis. Princeton University Press, Princeton, New Jersey.

Nott, M. P., and D. F. DeSante. 2002. Demographic monitoring and the identification of transients in mark-recapture models. *in* J. M. Scott, P. J. Heglund, M. Morrison, M. Raphael, J. Haufler, and B. Wall, editors. Predicting species occurrences: Issues of scale and accuracy. Island Press, Covello, California.

Oro, D., and D. F. Doak. 2020. Breeding transients in capture–recapture modeling and their consequences for local population dynamics. Scientific Reports 2020 10:1 10:1–13.

Plummer, M. 2003. JAGS: A Program for Analysis of Bayesian Graphical Models Using Gibbs Sampling. Proceedings of the 3rd International Workshop on Distributed Statistical Computing (DSC 2003), March 20-22, Vienna, Austria. ISSN 1609-395X.

Pollock, K. H., J. D. Nichols, C. Brownie, and J. E. Hines. 1990. Statistical inference for capture-recapture experiments. Wildlife Monographs:3–97.

Pradel, R. 1996. Utilization of capture-mark-recapture for the study of recruitment and population growth rate. Biometrics 52:703–709.

Pradel, R., J. E. Hines, J.-D. Lebreton, and J. D. Nichols. 1997. Capture-recapture survival models taking account of transients. Biometrics 53:60–72.

R Core Team. 2021. R: A language and environment for statistical computing. R Foundation for Statistical Computing, Vienna, Austria.

Robinson, R. A., R. Julliard, and J. F. Saracco. 2009. Constant effort: Studying avian population processes using standardised ringing. Ringing & Migration 24:199–204.

Royle, J. A. 2008. Modeling individual effects in the cormack–jolly–seber model: A state–space formulation. Biometrics 64:364–370.

Royle, J. A., and R. M. Dorazio. 2008. Hierarchical Modeling and Inference in Ecology. Page 444. Academic Press, London, UK.

Royle, J. A., and R. M. Dorazio. 2012. Parameter-expanded data augmentation for Bayesian analysis of capture–recapture models. Journal of Ornithology 152:521–537.

Ruiz-Gutierrez, V., W. L. Kendall, J. F. Saracco, and G. C. White. 2016. Overwintering strategies of migratory birds: a novel approach for estimating seasonal movement patterns of residents and transients. Journal of Applied Ecology 53:1035–1045.

Saracco, J. F., L. Helton, J. Liske-Clark, and P. Radley. 2020. Recent dynamics and trends of landbird populations on Saipan, Northern Mariana Islands. Pacific Science 74:319–329.

Saracco, J. F., J. A. Royle, D. F. DeSante, and B. Gardner. 2010. Modeling spatial variation in avian survival and residency probabilities. Ecology 91:1885–1891.

Saracco, J. F. and C. B. Yackulic. 2023. Replication data for "Fast and flexible Bayesian Jolly Seber models and application to populations with transients". Open Science Framework. https://doi.org/10.17605/OSF.IO/Q3458.

Sauer, J. R., J. E. Fallon, and R. Johnson. 2003. Use of North American Breeding Bird Survey data to estimate population change for bird conservation regions. Journal of Wildlife Management 67:372–389.

Schofield, M. R., and R. J. Barker. 2008. A unified capture-recapture framework. Journal of Agricultural, Biological, and Environmental Statistics 13:458.

Schwarz, C. J., and A. N. Arnason. 1996. A General Methodology for the Analysis of Capture-Recapture Experiments in Open Populations. Biometrics 52:860–873.

Seber, G. A. F. 1965. A note on the multiple recapture census. Biometrika 52:249–259.

Tenan, S., R. Pradel, G. Tavecchia, J. M. Igual, A. Sanz-Aguilar, M. Genovart, and D. Oro. 2014. Hierarchical modelling of population growth rate from individual capture–recapture data. Methods in Ecology and Evolution 5:606–614.

Tenan, S., G. Tavecchia, D. Oro, and R. Pradel. 2019. Assessing the effect of density on population growth when modeling individual encounter data. Ecology 100:e02595.

Wilson, S., J. F. Saracco, R. Krikun, D. T. T. Flockhart, C. M. Godwin, and K. R. Foster. 2018. Drivers of demographic decline across the annual cycle of a threatened migratory bird. Scientific Reports 8:7316.

Yackulic, C. B., M. Dodrill, M. Dzul, J. S. Sanderlin, and J. A. Reid. 2020. A need for speed in Bayesian population models: a practical guide to marginalizing and recovering discrete latent states. Ecological Applications 30:e02112.

Yackulic, C. B., M. D. Yard, J. Korman, and D. R. Van Haverbeke. 2014. A quantitative life history of endangered humpback chub that spawn in the Little Colorado River: Variation in movement, growth, and survival. Ecology and Evolution 4:1006–1018.

Table 1. 95% credible interval coverage and widths [mean with (SE) below] for the discrete latent state (D), marginalized (M), discrete latent state with 25% missing data (D.NA), and marginalized with 25% missing data (M.NA) multistate Jolly Seber models applied to 100 simulated data sets.

	% coverage					Width			
	D	M	D.NA	M.NA		D	M	D.NA	M.NA
$\boldsymbol{N_{super}}$	90 (3.0)	92 (2.7)	83 (3.8)	87 (3.4)		82.1 (0.9)	99.3 (0.7)	116.4 (1.7)	129 (1.3)
$\boldsymbol{\phi}$	94 (2.4)	94 (2.4)	92 (2.7)	91 (2.9)		0.104 (0.000)	0.104 (0.001)	0.132 (0.001)	0.132 (0.001)
$\boldsymbol{b_1}$	98 (1.4)	98 (1.4)	97 (1.7)	98 (1.4)		0.157 (0.001)	0.157 (0.001)	0.185 (0.001)	0.185 (0.001)
$\boldsymbol{b_2}$	97 (1.7)	97 (1.7)	97 (1.7)	97 (1.7)		0.010 (0.010)	0.010 (0.010)	0.010 (0.010)	0.010 (0.010)
$\boldsymbol{b_3}$	96 (2.0)	96 (2.0)	98 (1.4)	97 (1.7)		0.129 (0.001)	0.129 (0.001)	0.154 (0.002)	0.155 (0.002)
$\boldsymbol{b_4}$	94 (2.4)	94 (2.4)	96 (2.0)	97 (1.7)		0.123 (0.001)	0.124 (0.001)	0.150 (0.002)	0.150 (0.002)
$\boldsymbol{b_5}$	95 (2.2)	95 (2.2)	96 (2.0)	96 (2.0)		0.122 (0.001)	0.122 (0.001)	0.148 (0.002)	0.148 (0.002)
$\boldsymbol{b_6}$	95 (2.2)	95 (2.2)	97 (1.7)	97 (1.7)		0.119 (0.001)	0.119 (0.001)	0.146 (0.002)	0.146 (0.002)
$\boldsymbol{b_7}$	95 (2.2)	95 (2.2)	96 (2.0)	96 (2.0)		0.115 (0.001)	0.115 (0.001)	0.142 (0.002)	0.142 (0.002)
p	93 (2.6)	92 (2.7)	87 (3.4)	89 (3.1)		0.139 (0.001)	0.140 (0.001)	0.183 (0.002)	0.184 (0.002)

Figure 1. Boxplots of posterior medians for adult apparent survival probability, ϕ, entry probability, b, capture probability, p (a); annual abundance, N_t, and superpopulation abundance, N_{super}, (b) from discrete (D) and marginalized (M) multistate Jolly Seber models applied to 100 complete simulated data sets and 100 data sets with 25% missing data. Data-generating values are indicated with red lines.

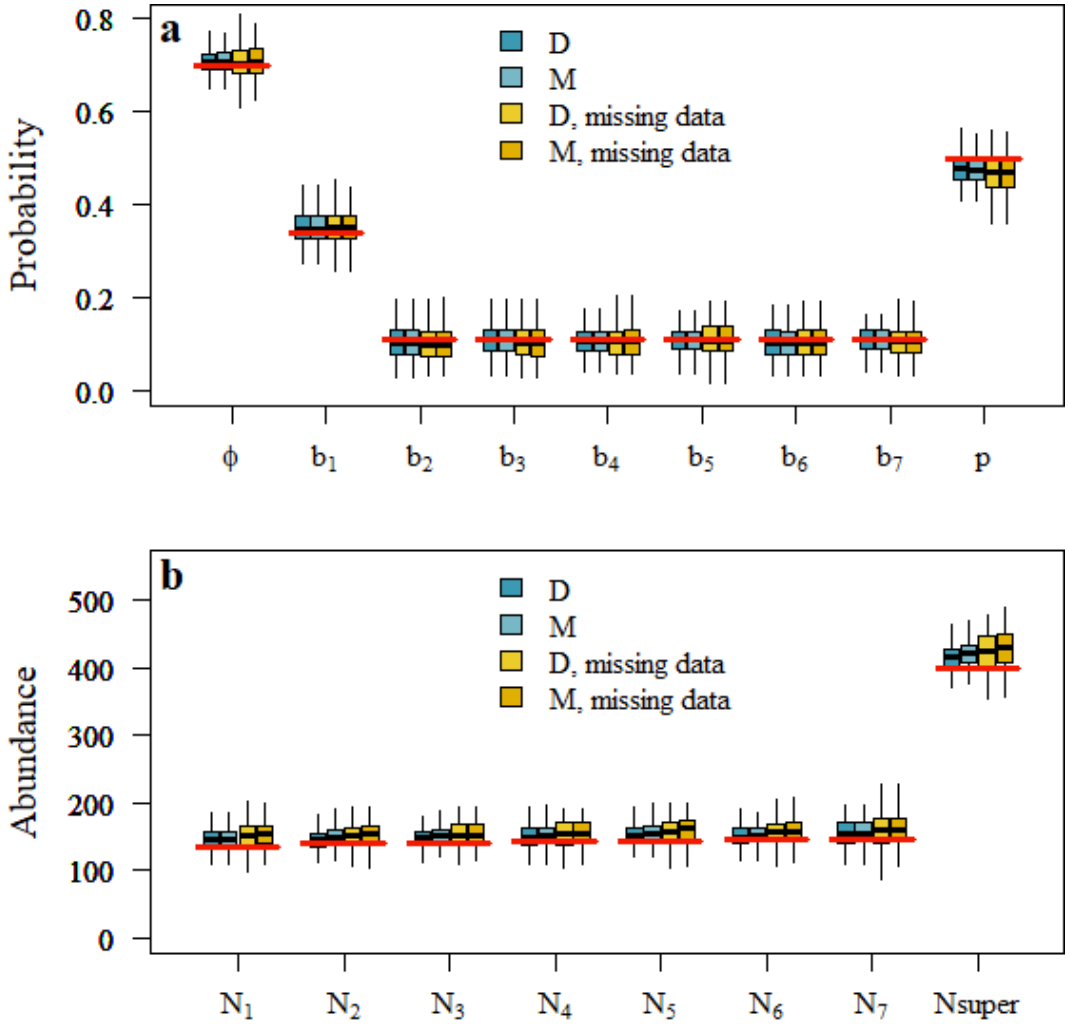


Figure 2. Boxplot of the time required to reach 1000 effective samples of the least converged parameter for discrete (D) and marginalized (M) multistate Jolly Seber models applied to 100 complete simulated data sets and to the same 100 data sets but with 25% of the data randomly missing.

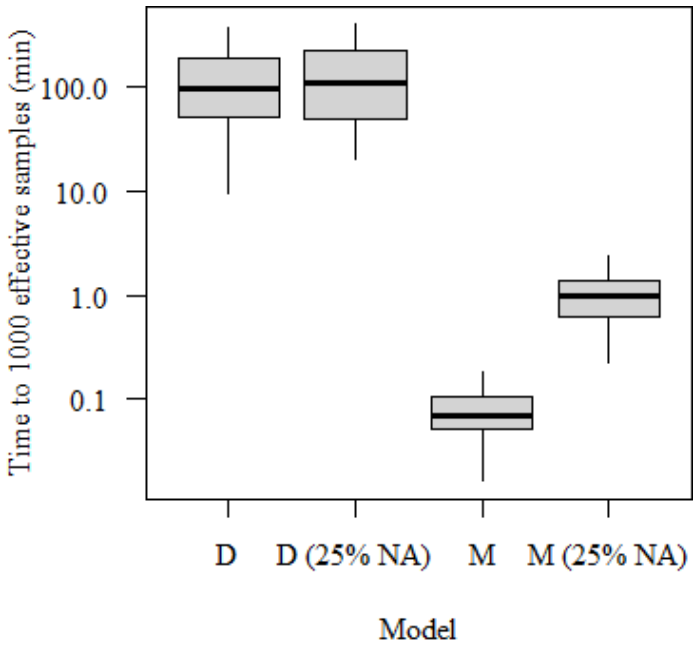


Figure 3. Boxplots of posterior medians for adult apparent survival probability, residency probability, entry probability, capture probability (a), and superpopulation size of resident individuals (b) from a marginalized multistate transient Jolly Seber model applied to 100 simulated data sets. Data-generating values are indicated with red lines. Mean 95% credible interval coverage of model parameters is shown in (c).

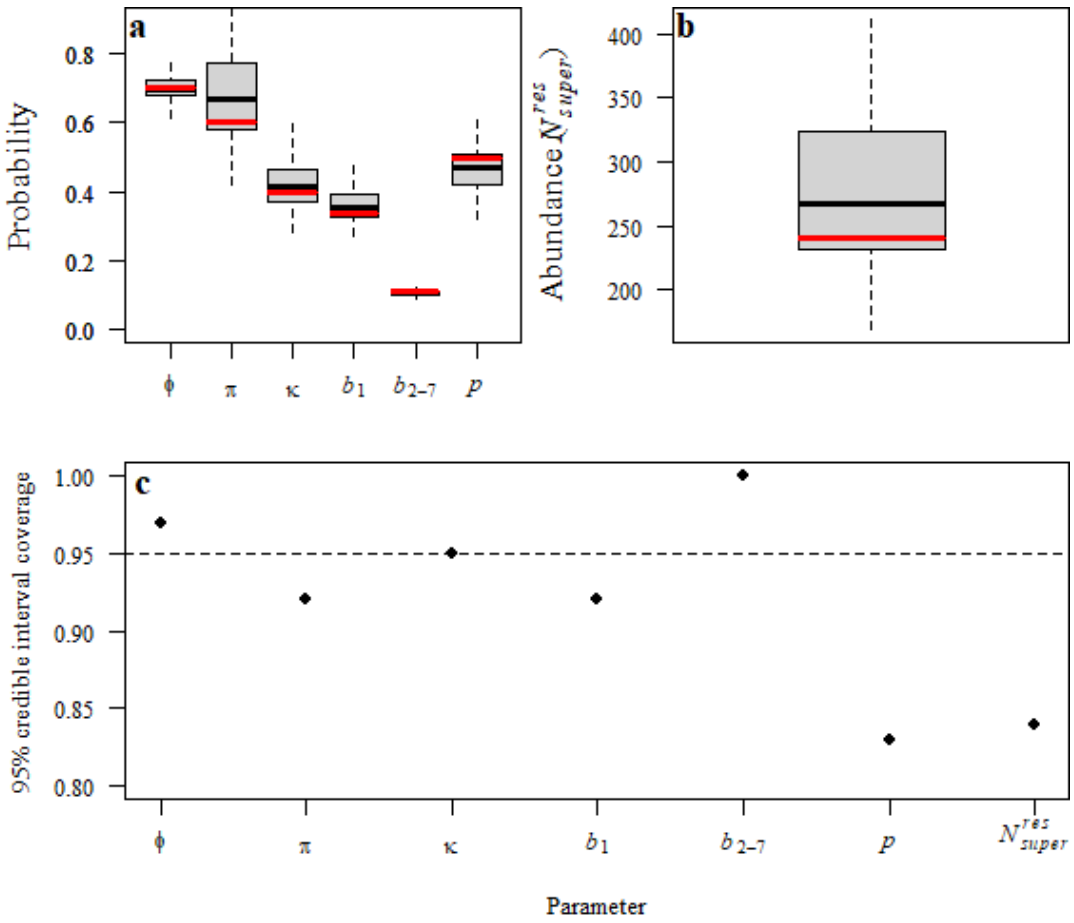
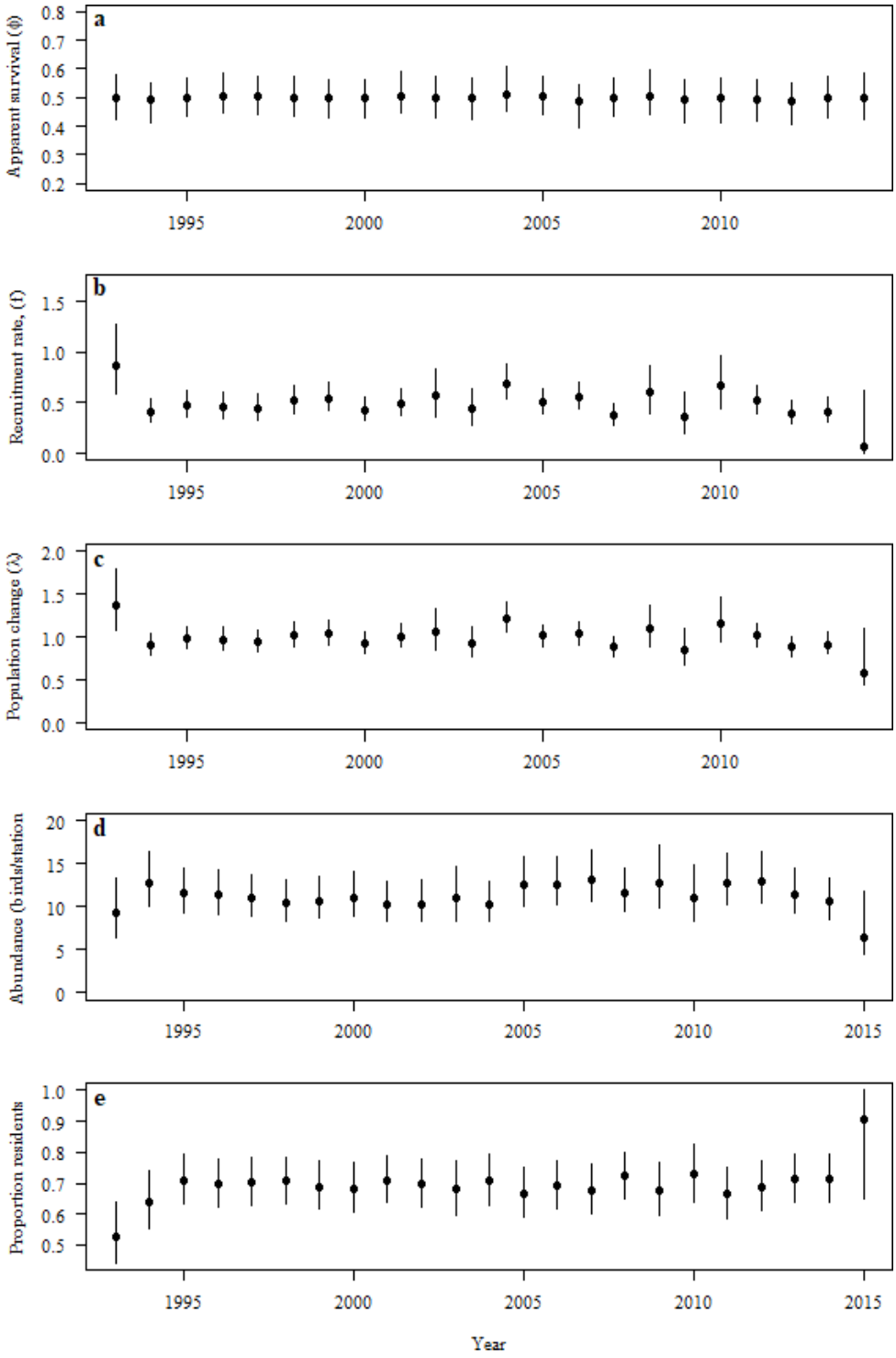


Figure 4. Medians ± 95% credible intervals for annual demographic parameters estimated for Kentucky warblers banded in the Central Hardwoods Bird Conservation Region as part of the Monitoring Avian Productivity and Survivorship (MAPS) program.



Supporting information for: Saracco, J. F. and C. B. Yackulic. Fast and flexible Bayesian Jolly Seber models and application to populations with transients

Appendix S1. BUGS/JAGS code for multistate Jolly Seber models.

In our simulation studies, we compared the discrete multistate JS model based on code in Kéry and Schaub (2011), Ch. 10, and compared results to a marginalized version of the same model. Here we provide model code in BUGS language, as implemented in popular software programs such as OpenBUGS (Lunn et al. 2009), JAGS (Plummer 2003), or NIMBLE (de Valpine et al. 2017) for these two model variations. Additional model detail is provided in the main manuscript.

First, priors for survival (ϕ), removal entry (γ), and capture probabilities (p) for both the discrete and marginalized models can be written as:

```
for (t in 1:(n.occasions-1)){
  phi[t] <- mean.phi
  gamma[t] ~ dunif(0, 1)
  p[t] <- mean.p
}

mean.phi ~ dunif(0, 1)
mean.p ~ dunif(0, 1)
```

Note that we follow Kéry and Schaub (2011) and Royle and Dorazio (2012) in parameterizing the recruitment process in terms of the removal entry probabilities, which can be related to the true entry probabilities, b_t, following:

$$b_1 = \gamma_1/\psi \text{ for } t = 1 \text{ and } b_t = \gamma_t \prod_{j=1}^{t-1} \left(1 - \gamma_j\right)/\psi \text{ for } t > 1,$$

where ψ represents the inclusion probability (i.e. probability that a member of the augmented data set is part of the superpopulation). In BUGS language, this can be written as:

```
for (t in 1:(n.occasions-1)){
    qgamma[t] <- 1-gamma[t]
```

```
    b[t] <- c[t]/psi
  }
  for (t in 2:(n.occasions-1)){
    c[t] <- gamma[t]*prod(qgamma[1:(t-1)])
  }
psi <- sum(c[1:(n.occasions-1)])
c[1] <- gamma[1]
```

Alternatively, we could define a model for b_t and derive the removal entry probabilities. The b_t could then be modeled as functions of covariates using a multinomial logit model (e.g., see Kéry and Schaub (2012), p. 301) or simply with a Dirichlet prior distribution. For example, as:

```
  b[1:(n.occasions-1)] ~ ddirich(alpha[1:(n.occasions-1)])
  psi ~ dunif (0, 1)
  gamma[1]<-psi*b[1]
 for (t in 2:n.occasions-1)){
    gamma[t] <- psi*b[t]/(1-sum(b[1:(t-1)])
    }
```

Second, state transition (**Ω**) and observation (**P**) arrays for both model types can be represented with:

```
#-----------------------------
# States
# 1 = not yet entered
# 2 = alive
# 3 = dead

# Observations
# 1 = captured
# 2 = not captured
#-----------------------------

# omega tracks probability of state at t+1 given state at t
  for (t in 1:(n.occasions-1)){
    omega[1,1,t] <- 1-gamma[t]
```

```
    omega[1,2,t] <- gamma[t]
    omega[1,3,t] <- 0
    omega[2,1,t] <- 0
    omega[2,2,t] <- phi[t]
    omega[2,3,t] <- 1-phi[t]
    omega[3,1,t] <- 0
    omega[3,2,t] <- 0
    omega[3,3,t] <- 1

# rho tracks capture probability at t given state at t
    rho[1,1,t] <- 0
    rho[1,2,t] <- 1
    rho[2,1,t] <- p[t]
    rho[2,2,t] <- 1 - p[t]
    rho[3,1,t] <- 0
    rho[3,2,t] <- 1
  }
```

Note that Kéry and Schaub (2011) also tracked state and observation probabilities across a fourth dimension of **Ω** and **P** representing individuals. This is unnecessary here, as parameters do not vary among individuals, and run time in JAGS (Plummer 2003) is approximately halved by not indexing by individuals. For the discrete latent state model, we then simulate the state ($s_{i,t}$) and conditional observations ($y_{i,t}$) from categorical distributions for all $i = 1, \ldots, M$ individuals at each time step:

```
  for (i in 1:M){
    s[i,1] ~ dcat(1)
    for (t in 2:n.occasions){
      s[i,t] ~ dcat(omega[s[i,t-1],1:3,t-1])
      y[i,t] ~ dcat(rho[s[i,t],1:2,t-1])
    }
  }
```

Note that the indexing for rho[,,t-1] here refers to capture probabilities at time t.

For the marginalized model, we loop through the $i = 1, \ldots, MS$ unique capture histories to track probabilities of state transitions and observations. Contributions of each unique

capture history likelihood to the overall likelihood were determined in the BUGS model using the modified “ones trick” described by Yackulic et al. (2020), whereby capture history frequencies are provided as data for both the frequency response variable fr and the number of trials FR in binomial distributions with success probabilities defined by the likelihoods:

```
  for (i in 1:MS){
    zeta[i,1,1] <- 1  # all individuals unrecruited at first dummy
occasion
    zeta[i,1,2] <- 0
    zeta[i,1,3] <- 0

    for (t in 2:n.occasions){
      zeta[i,t,1] <- inprod(zeta[i,(t-1),1:3], omega[1:3,1,(t-1)] *
rho[1,ys[i,t],(t-1)])
      zeta[i,t,2] <- inprod(zeta[i,(t-1),1:3], omega[1:3,2,(t-1)] *
rho[2,ys[i,t],(t-1)])
      zeta[i,t,3] <- inprod(zeta[i,(t-1),1:3], omega[1:3,3,(t-1)] *
rho[3,ys[i,t],(t-1)])
    }
    lik[i] <- sum(zeta[i,n.occasions,1:3])
    fr[i] ~ dbin(lik[i], FR[i])
  }
```

For the discrete latent state model we can then derive secondary parameters from first-order states following:

```
for (i in 1:M){
   for (t in 2:n.occasions){
      al[i,t-1] <- equals(s[i,t],2)
      } # t
   for (t in 1:(n.occasions-1)){
      d[i,t] <- equals(s[i,t]-al[i,t],0)
      } #t
   alive[i] <- sum(al[i,])
   } # i

```

```
for (t in 1:(n.occasions-1)){
   N[t] <- sum(al[,t])        # population size
   B[t] <- sum(d[,t])         # number of entries. Note that indexing in
paper
                              # goes from 0 (initial entry) to T
(n.occasions - 1)
   } # t
for (i in 1:M){
   w[i] <- 1-equals(alive[i],0) # w[i] = 1 if individual i ever entered
the
                                # population and 0 otherwise
   } # i
for(t in 1:(n.occasions-2)){
  lambda[t] <- N[t+1]/N[t] # population growth rate
  f[t] <- B[t+1]/N[t] # recruitment (per capita entry) rate. Could also be
                      # calculated as lambda[t] - phi[t]
} # t
Nsuper <- sum(w[]) # could also be calculated with sum(B[])
```

For the marginalized models, we can derive secondary parameters as functions of model parameters and first-order states:

```
  for (t in 1:(n.occasions-1)){
    qgamma[t] <- 1-gamma[t]
  } # t
  c[1] <- gamma[1]
  N[1] <- B[1]
  for (t in 2:(n.occasions-1)){
    c[t] <- gamma[t] * prod(qgamma[1:(t-1)])
    N[t] <- B[t] + phi[t] * N[t-1]
  } # t
  psi <- sum(c[1:(n.occasions-1)])    # inclusion probability
  for (t in 1:(n.occasions-1)){
    b[t] <- c[t] / psi      # entry probability
    B[t] <- c[t]*M          # number of entries
  } # t

```

```
for(t in 1:(n.occasions-2)){
lambda[t] <- N[t+1]/N[t] # population growth rate
f[t] <- B[t+1]/N[t]      # recruitment (per capita entry) rate
} # t

Nsuper <- sum(B[1:(n.occasions-1)]) # superpopulation size
```

Missing data models

While the code for the discrete latent state model does not change with missing data (which are just entered as "NA" values), the marginalized models require the following modifications. First, for capture probability we add an index for unique capture history and multiply by an indicator variable, $op.ind_{i,t}$ denoting whether the individuals represented by that capture history were observable ($op.ind_{i,t} = 1$) or not ($op.ind_{i,t} = 0$) during that time period:

```
for (t in 1:(n.occasions-1)){
```

⋮

```
  for (i in 1:MS){
  p[i,t] <- mean.p*op.ind[i,t] # op.ind: 1 = effort, 0 = no effort
  }
```

⋮

```
}
```

Second, for **P** we need to add a fourth dimension for the capture probability array:

```
for (t in 1:(n.occasions-1)){
```

⋮

```
  for (i in 1:MS){
  rho[1,1,t,i] <- 0
  rho[1,2,t,i] <- 1
  rho[2,1,t,i] <- p[i,t]
  rho[2,2,t,i] <- 1 - p[i,t]
```

```
    rho[3,1,t,i] <- 0
    rho[3,2,t,i] <- 1
    }
  }
```

References

de Valpine, P., D. Turek, C. Paciorek, C. Anderson-Bergman, D. Temple Lang, and R. Bodik. 2017. Programming with models: Writing statistical algorithms for general model structures with NIMBLE. Journal of Computational and Graphical Statistics 26:403–413.

Kéry, M., and M. Schaub. 2012. Bayesian Population Analysis using WinBUGS. Page 554. 1st Edition.

Lunn, D., D. Spiegelhalter, A. Thomas, and N. Best. 2009. The BUGS project: Evolution, critique and future directions. Statistics in Medicine 28:3049–3067.

Plummer, M. 2003. JAGS: A Program for Analysis of Bayesian Graphical Models Using Gibbs Sampling. Proceedings of the 3rd International Workshop on Distributed Statistical Computing (DSC 2003), March 20-22, Vienna, Austria. ISSN 1609-395X.

Royle, J. A., and R. M. Dorazio. 2012. Parameter-expanded data augmentation for Bayesian analysis of capture–recapture models. Journal of Ornithology 152:521–537.

Yackulic, C. B., M. Dodrill, M. Dzul, J. S. Sanderlin, and J. A. Reid. 2020. A need for speed in Bayesian population models: a practical guide to marginalizing and recovering discrete latent states. Ecological Applications 30:e02112.

Supporting information for: Saracco, J. and C. B. Yackulic. Efficient and flexible Bayesian Jolly Seber models and application to populations with transients

Appendix S2. Variations of multistate Jolly Seber models and methods for estimating derived quantities

We investigated four methods for estimating derived secondary parameters but reported only the first two approaches in the manuscript. These methods include:

1) Estimating parameters using model "d", the discrete latent state model presented in the manuscript. In this approach derived parameters are calculated directly from the posteriors of first-order states.

2) Estimating parameters using model "m1", the marginalized model presented in the manuscript. In this approach derived parameters are calculated as functions of modeled parameters (see main manuscript and Appendix S1 for details).

3) Estimating parameters using model "m2", which was not presented in the manuscript, but is identical to m1 except for in how derived parameters are estimated. In this approach, N_{super} is calculated by estimating the number of unobserved individuals by drawing from a binomial distribution using the probability of inclusion and the number of augmented individuals and this estimate of unobserved individuals is added to the known number of observed individuals. Other derived parameters are calculated by determining the full conditional estimates of different underlying dynamics based on the observed capture history and simulating feasible true recruit and death timing based on their relative likelihood based on an observed capture history.

4) Estimating parameters using model "m3," which was not presented in the manuscript, but is identical to m1 except for in how derived parameters are estimated. In this approach, N_{super} is calculated using the same approach as "m2". N_t is derived by calculating forward-conditional probabilities for each individual and time period and using a series of Bernoulli trials to determine the expected number of alive individuals. B's are calculated as functions of modelled parameters as in "m1".

We compared these four methods with respect to speed, coverage, and precision of estimates of derived parameters based on 100 simulated data sets. Below we share annotated code, as well as figures based on these simulations.

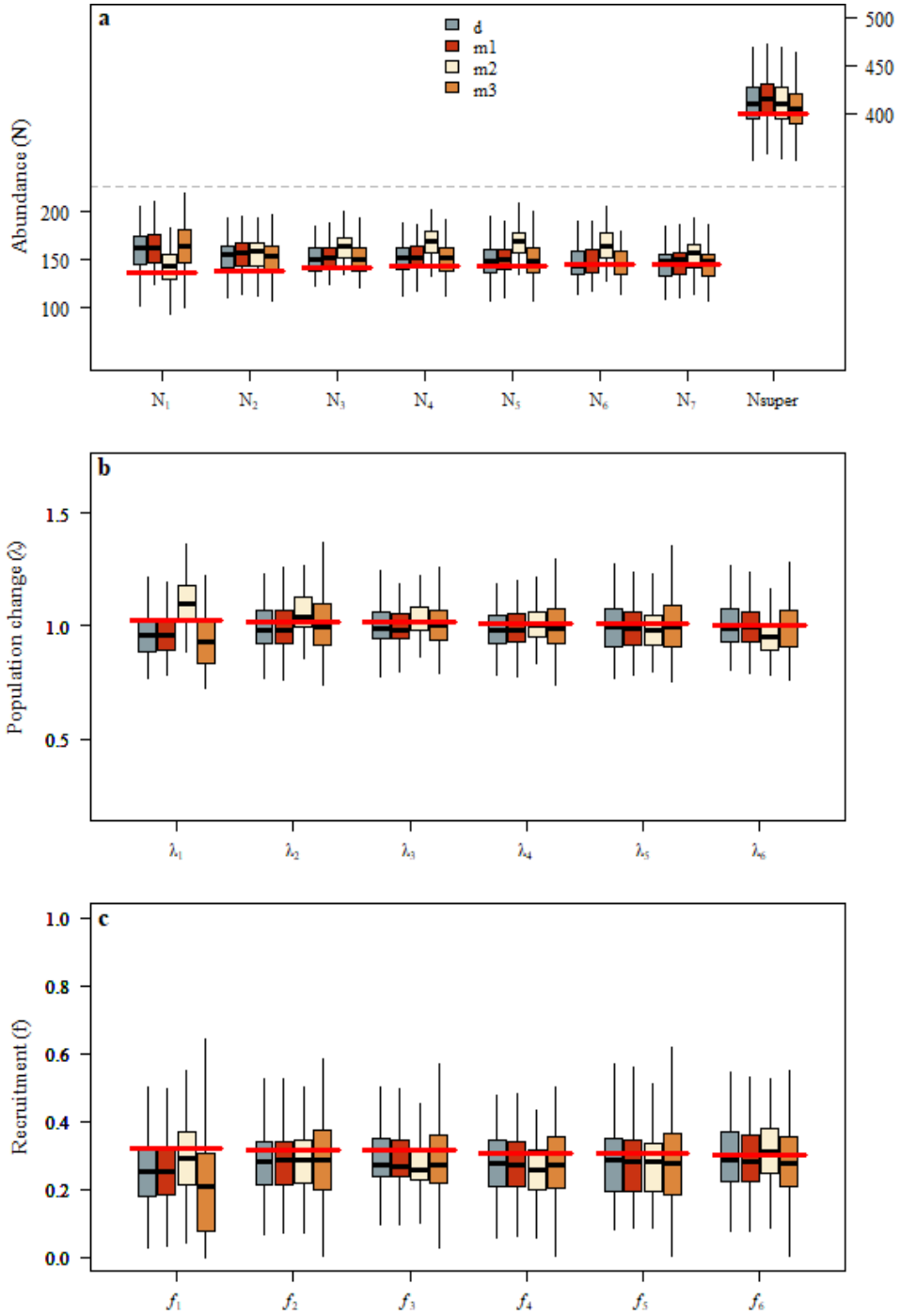
Credible interval coverage for the year-specific abundance estimates, N_t was relatively low for the discrete latent state model and especially the marginalized model m2 (Table S1). Credible interval coverage was highest and generally above the nominal 95% level for model m1, although this likely reflects the broader credible intervals for that model. For model m2, N_t estimates were biased high for time periods 3-7 (Fig. S1). Model m1 was fastest, but all of the marginalized models were relatively fast compared to model d (Fig. S2).

***Table S1**. 95% credible interval coverage and widths [mean (sd)] for discrete (d) and marginalized (m1, m2, and m3) JS models applied to 100 simulated data sets that differed with respect to how abundance (N), population change, and recruitment are estimated.*

	d		m1		m2		m3	
	% coverage	Width	% coverage	Width	% coverage	Width	% coverage	Width
N_{super}	90.0	80.4	93.0	97.0	89.0	80.7	90.0	77.2
	(30.2)	(11.4)	(25.6)	(8.5)	(31.4)	(10.6)	(30.2)	(9.7)
N_1	65.0	68.4	72.0	81.5	85.0	55.1	63.0	74.6
	(47.9)	(9.1)	(45.1)	(8.6)	(35.9)	(6.4)	(48.5)	(11.4)
N_2	72.0	54.7	85.0	67.3	60.0	50.7	77.0	57.5
	(45.1)	(7.4)	(35.9)	(6.8)	(49.2)	(5.8)	(42.3)	(9.0)
N_3	79.0	51.2	90.0	63.7	54.0	51.5	85.0	57.1
	(40.9)	(7.0)	(30.2)	(6.5)	(50.1)	(5.6)	(35.9)	(8.2)
N_4	85.0	50.0	93.0	63.6	48.0	53.6	88.0	57.7
	(35.9)	(6.6)	(25.6)	(6.1)	(50.2)	(5.7)	(32.7)	(7.8)
N_5	85.0	50.2	95.0	64.0	52.0	55.3	88.0	57.7
	(35.9)	(7.2)	(21.9)	(6.5)	(50.2)	(6.6)	(32.7)	(9.1)
N_6	91.0	52.0	98.0	65.4	65.0	57.2	91.0	57.7
	(28.8)	(7.1)	(14.1)	(6.6)	(47.9)	(7.0)	(28.8)	(8.6)
N_7	90.0	56.1	93.0	67.2	86.0	64.4	89.0	56.6
	(30.2)	(7.9)	(25.6)	(7.6)	(34.9)	(9.6)	(31.4)	(8.3)
λ_1	84.0	0.403	89.0	0.451	87.0	0.406	79.0	0.429

	d		m1		m2		m3	
	% coverage	Width	% coverage	Width	% coverage	Width	% coverage	Width
	(36.8)	(0.078)	(31.4)	(0.098)	(33.8)	(0.072)	(40.9)	(0.087)
λ_2	90.0	0.367	92.0	0.410	88.0	0.314	84.0	0.412
	(30.2)	(0.057)	(27.3)	(0.071)	(32.7)	(0.042)	(36.8)	(0.063)
λ_3	93.0	0.353	95.0	0.392	90.0	0.281	93.0	0.401
	(25.6)	(0.043)	(21.9)	(0.051)	(30.2)	(0.029)	(25.6)	(0.047)
λ_4	87.0	0.339	91.0	0.374	87.0	0.260	90.0	0.386
	(33.8)	(0.050)	(28.8)	(0.060)	(33.8)	(0.029)	(30.2)	(0.055)
λ_5	88.0	0.343	90.0	0.376	80.0	0.253	88.0	0.390
	(32.7)	(0.051)	(30.2)	(0.062)	(40.2)	(0.026)	(32.7)	(0.058)
λ_6	89.0	0.354	92.0	0.385	85.0	0.268	87.0	0.390
	(31.4)	(0.048)	(27.3)	(0.061)	(35.9)	(0.025)	(33.8)	(0.052)
f_1	85.0	0.416	91.0	0.458	90.0	0.445	78.0	0.445
	(35.9)	(0.086)	(28.8)	(0.103)	(30.2)	(0.080)	(41.6)	(0.137)
f_2	90.0	0.369	93.0	0.414	89.0	0.342	88.0	0.433
	(30.2)	(0.059)	(25.6)	(0.074)	(31.4)	(0.045)	(32.7)	(0.083)

	d		m1		m2		m3	
	% coverage	Width	% coverage	Width	% coverage	Width	% coverage	Width
f_3	93.0	0.349	97.0	0.397	89.0	0.307	92.0	0.418
	(25.6)	(0.044)	(17.1)	(0.053)	(31.4)	(0.032)	(27.3)	(0.058)
f_4	88.0	0.332	92.0	0.379	85.0	0.290	89.0	0.393
	(32.7)	(0.052)	(27.3)	(0.065)	(35.9)	(0.036)	(31.4)	(0.074)
f_5	87.0	0.333	92.0	0.383	85.0	0.302	88.0	0.393
	(33.8)	(0.053)	(27.3)	(0.066)	(35.9)	(0.038)	(32.7)	(0.073)
f_6	90.0	0.342	92.0	0.393	86.0	0.326	89.0	0.392
	(30.2)	(0.051)	(27.3)	(0.064)	(34.9)	(0.034)	(31.4)	(0.063)



872

873 ***Figure S1***. *Boxplots of posterior medians of derived parameters from discrete (d) and*
874 *marginalized (m1, m2, m3) multistate Jolly Seber models applied to 100 simulated data*
875 *sets. Abundance estimates are shown in (a; note right y-axis shows scale for* N_{super}*);*
876 *population change in (b) and recruitment in (c). Data-generating values are indicated*
877 *with red lines.*

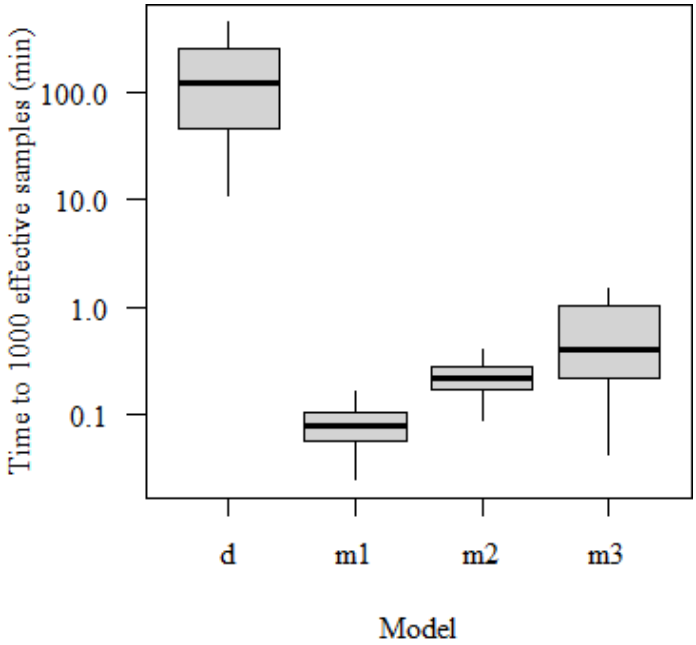


Figure S2. *Boxplot of the time required to reach 1000 effective samples of the least converged parameter for discrete (d) and marginalized (m1, m2, and m3) multistate Jolly Seber models applied to 100 complete simulated data sets.*

R and JAGS code for writing model files, simulating data, running models, and extracting results, are as follows:

```
#-----------------------------------------------------------------------
---# Write models to files

# 1. Discrete latent state model (d)
{
sink("js-ms-d.txt")
cat("
model {

#-------------------------------------
# phi: survival probability
# gamma: removal entry probability
# p: capture probability
#-------------------------------------

# Priors and constraints
for (t in 1:(n.occasions-1)){
   phi[t] <- mean.phi     # Constant survival
   gamma[t] ~ dunif(0,1)  # Time-dependent entry
   p[t] <- mean.p         # Constant capture probability
   }

mean.phi ~ dunif(0,1)    # Prior for mean survival
mean.p ~ dunif(0,1)      # Prior for mean capture
```

```
#-------------------------------------
# States (S):
# 1 not yet entered
# 2 alive
# 3 dead
#-------------------------------------

# Probabilities of state S(t+1) given S(t)
   for (t in 1:(n.occasions-1)){
      omega[1,1,t] <- 1-gamma[t]
      omega[1,2,t] <- gamma[t]
      omega[1,3,t] <- 0
      omega[2,1,t] <- 0
      omega[2,2,t] <- phi[t]
      omega[2,3,t] <- 1-phi[t]
      omega[3,1,t] <- 0
      omega[3,2,t] <- 0
      omega[3,3,t] <- 1

#-------------------------------------
# Observations (O):
# 1 captured
# 2 not captured
#-------------------------------------

# Probabilities of O(t) given S(t)
      rho[1,1,t] <- 0
      rho[1,2,t] <- 1
      rho[2,1,t] <- p[t]
      rho[2,2,t] <- 1-p[t]
      rho[3,1,t] <- 0
      rho[3,2,t] <- 1
      } #t

# Likelihood
for (i in 1:M){
   # Define latent state at first occasion
   s[i,1] ~ dcat(1)   # Make sure that all M individuals are in state 1 at
t=1
   for (t in 2:n.occasions){
      # State process: draw S(t) given S(t-1)
      s[i,t] ~ dcat(omega[s[i,t-1],,t-1])
      # Observation process: draw O(t) given S(t)
      y[i,t] ~ dcat(rho[s[i,t],,t-1])
      } #t
   } #i

# Derived population parameters
for (t in 1:(n.occasions-1)){
   qgamma[t] <- 1-gamma[t]
   }
c[1] <- gamma[1]
```

```
for (t in 2:(n.occasions-1)){
   c[t] <- gamma[t] * prod(qgamma[1:(t-1)])
   } # t
psi <- sum(c[1:(n.occasions-1)])            # inclusion probability
for (t in 1:(n.occasions-1)){
   b[t] <- c[t] / psi      # entry probability
   } # t

for (i in 1:M){
   for (t in 2:n.occasions){
      al[i,t-1] <- equals(s[i,t],2)
      } # t
   for (t in 1:(n.occasions-1)){
      d[i,t] <- equals(s[i,t]-al[i,t],0)
      } #t
   alive[i] <- sum(al[i,])
   } # i

for (t in 1:(n.occasions-1)){
   N[t] <- sum(al[,t])        # population size
   B[t] <- sum(d[,t])         # number of entries
   } # t
for (i in 1:M){
   w[i] <- 1-equals(alive[i],0)
   } # i
for(t in 1:(n.occasions-2)){
  lambda[t] <- N[t+1]/N[t] # population growth rate
  f1[t] <- B[t+1]/N[t] # recruitment rate
  f2[t] <- lambda[t] - phi[t]
} # t
Nsuper <- sum(w[])            # superpopulation size
}
",fill = TRUE)
sink()


# 2. Marginalized multistate Jolly Seber model - derived parameters (m1)

sink("js-ms-m1.txt")
cat("

model{

  # Priors and constraints
  for (t in 1:(n.occasions-1)){
    phi[t] <- mean.phi     # Constant survival
    gamma[t] ~ dunif(0,1)  # Time-dependent entry probabilities
    p[t] <- mean.p         # Constant capture probability
  }

  mean.phi ~ dunif(0,1)    # Prior for mean survival
  mean.p ~ dunif(0,1)      # Prior for mean capture

```

```
  for (t in 1:(n.occasions-1)){

  #-------------------------------------------------------
  # states (S)
  # 1 = not yet entered
  # 2 = alive
  # 3 = dead
  #-------------------------------------------------------

  # Probabilities of state S(t+1) given S(t)
    omega[1,1,t] <- 1-gamma[t]
    omega[1,2,t] <- gamma[t]
    omega[1,3,t] <- 0
    omega[2,1,t] <- 0
    omega[2,2,t] <- phi[t]
    omega[2,3,t] <- 1-phi[t]
    omega[3,1,t] <- 0
    omega[3,2,t] <- 0
    omega[3,3,t] <- 1

  #-------------------------------------------------------
  # observations (O)
  # 1 = captured
  # 2 = not captured
  #-------------------------------------------------------

  # Probabilities of O(t) given S(t)
    rho[1,1,t] <- 0
    rho[1,2,t] <- 1
    rho[2,1,t] <- p[t]
    rho[2,2,t] <- 1 - p[t]
    rho[3,1,t] <- 0
    rho[3,2,t] <- 1

  } # t

  # Likelihood
  for (i in 1:MS){
    # Define latent state at first occasion
    zeta[i,1,1] <- 1   # Set all individuals to state 1 (not yet entered)
at t=1
    zeta[i,1,2] <- 0
    zeta[i,1,3] <- 0

    for (t in 2:(n.occasions)){
      # State process: draw S(t+1) given S(t)
      zeta[i,t,1] <- inprod(zeta[i,(t-1),1:3], omega[1:3,1,(t-1)] *
      rho[1,ys[i,t],(t-1)])
      zeta[i,t,2] <- inprod(zeta[i,(t-1),1:3], omega[1:3,2,(t-1)] *
      rho[2,ys[i,t],(t-1)])
      zeta[i,t,3] <- inprod(zeta[i,(t-1),1:3], omega[1:3,3,(t-1)] *
      rho[3,ys[i,t],(t-1)])
    } # t
```

```
    lik[i] <- sum(zeta[i,(n.occasions),1:3])
    fr[i] ~ dbin(lik[i], FR[i])
  } # i

  # Calculate derived population parameters
  for (t in 1:(n.occasions-1)){
    qgamma[t] <- 1-gamma[t]
  } # t
  c[1] <- gamma[1]
  N[1] <- B[1]
  for (t in 2:(n.occasions-1)){
    c[t] <- gamma[t] * prod(qgamma[1:(t-1)])
    N[t] <- B[t] + phi[t] * N[t-1]
  } # t
  psi <- sum(c[1:(n.occasions-1)])    # inclusion probability
  for (t in 1:(n.occasions-1)){
    b[t] <- c[t] / psi      # entry probability
    B[t] <- c[t]*M       # number of entries
  } # t

  for(t in 1:(n.occasions-2)){
  lambda[t] <- N[t+1]/N[t] # population growth rate
  f1[t] <- B[t+1]/N[t] # recruitment rate
  f2[t] <- lambda[t] - phi[t]
  } # t

  Nsuper <- sum(B[1:(n.occasions-1)]) # superpopulation size
}

",fill=TRUE)
sink()

# 3. Marginalized multistate Jolly Seber model - simulated N and B (m2)

sink("js-ms-m2.txt")
cat("
model{

  # Priors and constraints
  for (t in 1:(n.occasions-1)){
    phi[t] <- mean.phi     # Constant survival
    gamma[t] ~ dunif(0,1)  # Time-dependent entry probabilities
    p[t] <- mean.p         # Constant capture probability
  }

  mean.phi ~ dunif(0,1)    # Prior for mean survival
  mean.p ~ dunif(0,1)      # Prior for mean capture

  for (t in 1:(n.occasions-1)){

  #-------------------------------------------------------
  # states (S)
  # 1 = not yet entered
```

```
  # 2 = alive
  # 3 = dead
  #-------------------------------------------------------

  # Probabilities of state S(t+1) given S(t)
    omega[1,1,t] <- 1-gamma[t]
    omega[1,2,t] <- gamma[t]
    omega[1,3,t] <- 0
    omega[2,1,t] <- 0
    omega[2,2,t] <- phi[t]
    omega[2,3,t] <- 1-phi[t]
    omega[3,1,t] <- 0
    omega[3,2,t] <- 0
    omega[3,3,t] <- 1

  #-------------------------------------------------------
  # observations (O)
  # 1 = captured
  # 2 = not captured
  #-------------------------------------------------------

  # Probabilities of O(t) given S(t)
    rho[1,1,t] <- 0
    rho[1,2,t] <- 1
    rho[2,1,t] <- p[t]
    rho[2,2,t] <- 1 - p[t]
    rho[3,1,t] <- 0
    rho[3,2,t] <- 1

  } # t

  # Likelihood
  for (i in 1:MS){
    # Define latent state at first occasion
    zeta[i,1,1] <- 1   # Set all individuals to state 1 (not yet entered)
at t=1
    zeta[i,1,2] <- 0
    zeta[i,1,3] <- 0

    for (t in 2:(n.occasions)){
      # State process: draw S(t+1) given S(t)
      zeta[i,t,1] <- inprod(zeta[i,(t-1),1:3], omega[1:3,1,(t-1)] *
      rho[1,ys[i,t],(t-1)])
      zeta[i,t,2] <- inprod(zeta[i,(t-1),1:3], omega[1:3,2,(t-1)] *
      rho[2,ys[i,t],(t-1)])
      zeta[i,t,3] <- inprod(zeta[i,(t-1),1:3], omega[1:3,3,(t-1)] *
      rho[3,ys[i,t],(t-1)])
    } # t
    lik[i] <- sum(zeta[i,(n.occasions),1:3])
    fr[i] ~ dbin(lik[i], FR[i])
  } # i

  # Calculate derived population parameters
```

```
  for (t in 1:(n.occasions-1)){
    qgamma[t] <- 1-gamma[t]
  } # t
  c[1] <- gamma[1]
  for (t in 2:(n.occasions-1)){
    c[t] <- gamma[t] * prod(qgamma[1:(t-1)])
  } # t
  psi <- sum(c[1:(n.occasions-1)])    # inclusion probability
  for (t in 1:(n.occasions-1)){
    b[t] <- c[t] / psi                # entry probability
  } # t

  p0_inc<-sum(zeta[MS,n.occasions,2:3])/sum(zeta[MS,n.occasions,1:3])
  U ~ dbin(p0_inc,FR[MS]) #assumes empty CH is always the last one
  Nsuper <- sum(FR[1:(MS-1)])+U # superpopulation size

# This loop calculates the likelihood that an individual recruits during
time
# step poss[i,1], dies during time step poss[i,2] and is never seen while
alive

for (i in 1:nposs){
    lik0[i]<-omega[1,2,poss[i,1]]*((1-equals(poss[i,2],(n.occasions-1)))*

(prod(rho[2,2,poss[i,1]:poss[i,2]])*prod(omega[2,2,poss[i,1]:(n.occasions-
1)])*
    (omega[2,3,poss[i,2]])/prod(omega[2,2,(poss[i,2]):(n.occasions-1)]))+
        (equals(poss[i,2],(n.occasions-
1)))*(prod(rho[2,2,poss[i,1]:poss[i,2]])
        *prod(omega[2,2,poss[i,1]:(n.occasions-
1)])/prod(omega[2,2,poss[i,2]:(n.occasions-1)])))
}

# The next set of loops determines for each individual (linked to
summarized
# capture history by lookup) determine if an underlying set of underlying
dynamics
# (i.e., recruit and death period) is possible given their capture history

for (i in 1:Nobs_ind){
    for (j in 1:(n.occasions-1)){
        for (k in 1:nposs){
            inc_a[i,j,k]<-
equals(poss[k,1],j)*equals(poss_obs[k,lookup[i]],1)
            inc_l[i,j,k]<-
equals(poss[k,2],j)*equals(poss_obs[k,lookup[i]],1)
            }
        p_a[i,j]<-inprod(inc_a[i,j,1:nposs],lik0[1:nposs])
        p_l[i,j]<-inprod(inc_l[i,j,1:nposs],lik0[1:nposs])
        }
    ar[i]~dcat(p_a[i,])
    li[i]~dcat(p_l[i,])
    A[i]<-ar[i]
```

```
    L[i]<-li[i]
    for (j in 1:(n.occasions-1)){
        S[i,j]<-step(j-A[i])*step(L[i]-j)
        }
}

# The next set of loops determines the likelihood of an underlying set of
# underlying dynamics (i.e., recruit and death period) given an individual
# was never seen.

for (j in 1:(n.occasions-1)){
    for (k in 1:nposs){
        inc_a_no[j,k]<-equals(poss[k,1],j)
        inc_l_no[j,k]<-equals(poss[k,2],j)
        }
    p_a_no[j]<-inprod(inc_a_no[j,1:nposs],lik0[1:nposs])
    p_l_no[j]<-inprod(inc_l_no[j,1:nposs],lik0[1:nposs])
    }

for (i in (Nobs_ind+1):(Npos_ind)){
    ar[i]~dcat(p_a_no)
    li[i]~dcat(p_l_no)
    A[i]<-ar[i]*step(Nobs_ind+U-i)
    L[i]<-li[i]*step(Nobs_ind+U-i)
    for (j in 1:(n.occasions-1)){
        S[i,j]<-step(j-A[i])*step(L[i]-j)
        }
    }

for (t in 1:(n.occasions-1)){
  B[t] <- sum(equals(A[1:(Npos_ind)],t)) # number of entries
  N[t] <- sum(S[1:(Npos_ind),t])
  } # t
  for(t in 1:(n.occasions-2)){
  lambda[t] <- N[t+1]/N[t] # population growth rate
  f1[t] <- B[t+1]/N[t]# recruitment rate
  f2[t] <- lambda[t] - phi[t]
  } # t
 }
    ", fill =TRUE)
sink()

# 3. Marginalized multistate Jolly Seber model - simulated N, not B (m3)

sink("js-ms-m3.txt")
cat("
model{

  # Priors and constraints
  for (t in 1:(n.occasions-1)){
    phi[t] <- mean.phi     # Constant survival
    gamma[t] ~ dbeta(1/(n.occasions-1), 2-t/(n.occasions-1)) # Time-
dependent entry probabilities
```

```
    p[t] <- mean.p         # Constant capture probability
  }

  mean.phi ~ dunif(0,1)    # Prior for mean survival
  mean.p ~ dunif(0,1)      # Prior for mean capture

  for (t in 1:(n.occasions-1)){

  #-------------------------------------------------------
  # states (S)
  # 1 = not yet entered
  # 2 = alive
  # 3 = dead
  #-------------------------------------------------------

  # Probabilities of state S(t+1) given S(t)
    omega[1,1,t] <- 1-gamma[t]
    omega[1,2,t] <- gamma[t]
    omega[1,3,t] <- 0
    omega[2,1,t] <- 0
    omega[2,2,t] <- phi[t]
    omega[2,3,t] <- 1-phi[t]
    omega[3,1,t] <- 0
    omega[3,2,t] <- 0
    omega[3,3,t] <- 1

  #-------------------------------------------------------
  # observations (O)
  # 1 = captured
  # 2 = not captured
  #-------------------------------------------------------

  # Probabilities of O(t) given S(t)
    rho[1,1,t] <- 0
    rho[1,2,t] <- 1
    rho[2,1,t] <- p[t]
    rho[2,2,t] <- 1 - p[t]
    rho[3,1,t] <- 0
    rho[3,2,t] <- 1

  } # t

  # Likelihood
  for (i in 1:MS){
    # Define latent state at first occasion
    zeta[i,1,1] <- 1   # Set all individuals to state 1 (not yet entered)
at t=1
    zeta[i,1,2] <- 0
    zeta[i,1,3] <- 0

    for (t in 2:(n.occasions)){
      zeta[i,t,1] <- inprod(zeta[i,(t-1),1:3], omega[1:3,1,(t-1)] *
      rho[1,ys[i,t],(t-1)])
```

```
      zeta[i,t,2] <- inprod(zeta[i,(t-1),1:3], omega[1:3,2,(t-1)] *
      rho[2,ys[i,t],(t-1)])
      zeta[i,t,3] <- inprod(zeta[i,(t-1),1:3], omega[1:3,3,(t-1)] *
      rho[3,ys[i,t],(t-1)])
    } # t
    lik[i] <- sum(zeta[i,(n.occasions),1:3])
    fr[i] ~ dbin(lik[i], FR[i])
  } # i

  # Calculate derived population parameters
  for (t in 1:(n.occasions-1)){
    qgamma[t] <- 1-gamma[t]
  } # t
  c[1] <- gamma[1]
  for (t in 2:(n.occasions-1)){
    c[t] <- gamma[t] * prod(qgamma[1:(t-1)])
  } # t
  psi <- sum(c[1:(n.occasions-1)])    # inclusion probability
  for (t in 1:(n.occasions-1)){
    b[t] <- c[t] / psi                # entry probability
    B[t] <- c[t]*M
  } # t

  p0_inc<-sum(zeta[MS,n.occasions,2:3])/sum(zeta[MS,n.occasions,1:3])
  U ~ dbin(p0_inc,FR[MS]) #assumes empty CH is always the last one
  Nsuper <- sum(FR[1:(MS-1)])+U # superpopulation size

  for (i in 1:M){
  # Simulations need to be done for individuals even though zeta can be
calculated
  # for summarized capture histories. lookup refers each individual in M,
to the
  # appropriate summarized capture history.
  for (t in 2:(n.occasions)){
  al[i,t-1] ~ dbern(zeta[lookup[i],t,2]/sum(zeta[lookup[i],t,1:3]))
  } # t
  } # i

  for (t in 1:(n.occasions-1)){
  N[t] <- sum(al[1:M,t]) # population size
  } # t

  for(t in 1:(n.occasions-2)){
  lambda[t] <- N[t+1]/N[t] # population growth rate

  f1[t] <- B[t+1]/N[t]# recruitment rate
  f2[t] <- lambda[t] - phi[t] # recruitment rate
  } # t
 }
    ", fill =TRUE)
sink()
}
```

```r
# some functions -------------------------------------------------------
---

# Simulate capture histories (from Kery and Schaub 2011)

simul.js <- function(PHI, P, b, Nsuper){
  B <- rmultinom(1, Nsuper, b) # Generate no. of entering ind. per
occasion
  n.occasions <- dim(PHI)[2] + 1
  CH.sur <- CH.p <- matrix(0, ncol = n.occasions, nrow = Nsuper)
  # Define a vector with the occasion of entering the population
  ent.occ <- numeric()
  for (t in 1:n.occasions){
    ent.occ <- c(ent.occ, rep(t, B[t]))
  }
  # Simulating survival
  for (i in 1:Nsuper){
    CH.sur[i, ent.occ[i]] <- 1   # Write 1 when ind. enters the pop.
    if (ent.occ[i] == n.occasions) next
    for (t in (ent.occ[i]+1):n.occasions){
      # Bernoulli trial: has individual survived occasion?
      sur <- rbinom(1, 1, PHI[i,t-1])
      ifelse (sur==1, CH.sur[i,t] <- 1, break)
    } #t
  } #i
  # Simulating capture
  for (i in 1:Nsuper){
    CH.p[i,] <- rbinom(n.occasions, 1, P[i,])
  } #i
  # Full capture-recapture matrix
  CH <- CH.sur * CH.p

  # Remove individuals never captured
  cap.sum <- rowSums(CH)
  never <- which(cap.sum == 0)
  CH <- CH[-never,]
  Nt <- colSums(CH.sur)    # Actual population size
  return(list(CH=CH, B=B, Nsuper=Nt))
}


# Collapse capture histories (from Yackulic et al. 2020)
collapse.ch <- function(ch){
  ch.char = apply(ch, 1, function(x) paste(x, collapse = ","))

  ch.sum.out = t(sapply(strsplit(names(table(ch.char)), split = ","),
                        as.numeric))
  fr.out = as.numeric(as.vector(table(ch.char)))

  return(list(ch.sum.out, fr.out))
}
```

```
#-----------------------------------------------------------------------
---

library(jagsUI)
library(dplyr)

##
## Attaching package: 'dplyr'

## The following objects are masked from 'package:stats':
##
##     filter, lag

## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union

library(data.table)

##
## Attaching package: 'data.table'

## The following objects are masked from 'package:dplyr':
##
##     between, first, last

# some functions -------------------------------------------------------
------

# Simulate capture histories (from Kery and Schaub 2011)

simul.js <- function(PHI, P, b, Nsuper){
  B <- rmultinom(1, Nsuper, b) # Generate no. of entering ind. per
occasion
  n.occasions <- dim(PHI)[2] + 1
  CH.sur <- CH.p <- matrix(0, ncol = n.occasions, nrow = Nsuper)
  # Define a vector with the occasion of entering the population
  ent.occ <- numeric()
  for (t in 1:n.occasions){
    ent.occ <- c(ent.occ, rep(t, B[t]))
  }
  # Simulating survival
  for (i in 1:Nsuper){
    CH.sur[i, ent.occ[i]] <- 1   # Write 1 when ind. enters the pop.
    if (ent.occ[i] == n.occasions) next
    for (t in (ent.occ[i]+1):n.occasions){
      # Bernoulli trial: has individual survived occasion?
      sur <- rbinom(1, 1, PHI[i,t-1])
      ifelse (sur==1, CH.sur[i,t] <- 1, break)
    } #t
  } #i
  # Simulating capture
  for (i in 1:Nsuper){
    CH.p[i,] <- rbinom(n.occasions, 1, P[i,])
  } #i
```

```
  # Full capture-recapture matrix
  CH <- CH.sur * CH.p

  # Remove individuals never captured
  cap.sum <- rowSums(CH)
  never <- which(cap.sum == 0)
  CH <- CH[-never,]
  Nt <- colSums(CH.sur)    # Actual population size
  return(list(CH=CH, B=B, Nsuper=Nt))
}


# Fill in initial values for s. Set to 1 (not entered) for all occasions
prior to
# first capture, 2 for occasions from first to last capture, and 3 for
occasions
# after last capture (from Kery and Schaub 2011).

js.multistate.init <- function(ch, ns){
  ch[ch==2] <- NA
  state <- ch
  for (i in 1:nrow(ch)){
    n1 <- min(which(ch[i,]==1))
    n2 <- max(which(ch[i,]==1))
    state[i,n1:n2] <- 2
  }
  state[state==0] <- NA
  get.first <- function(x) min(which(!is.na(x)))
  get.last <- function(x) max(which(!is.na(x)))
  f <- apply(state, 1, get.first)
  l <- apply(state, 1, get.last)
  for (i in 1:nrow(ch)){
    state[i,1:f[i]] <- 1
    if(l[i]!=ncol(ch)) state[i, (l[i]+1):ncol(ch)] <- 3
    state[i, f[i]] <- 2
  }
  state <- rbind(state, matrix(1, ncol = ncol(ch), nrow = ns))
  return(state)
}

# Collapse capture histories (from Yackulic et al. 2020)

collapse.ch <- function(ch){
  options(warn=-1)
  ch.char = apply(ch, 1, function(x) paste(x, collapse = ","))
  ch.sum.out = t(sapply(strsplit(names(table(ch.char)), split = ","),
as.numeric))
  fr.out = as.numeric(as.vector(table(ch.char)))
  return(list(ch.sum.out, fr.out))
}

nrep <- 100
# Define parameter values
```

```
n.occasions <- 7                          # Number of capture occasions
Nsuper <- 400                             # Superpopulation size
phi <- rep(0.7, n.occasions-1)            # Survival probabilities
b <- c(0.40, rep(0.11, n.occasions-1))    # Entry probabilities
p <- rep(0.5, n.occasions)                # Capture probabilities

PHI <- matrix(rep(phi, (n.occasions-1)*Nsuper), ncol = n.occasions-1, nrow
=
                Nsuper, byrow = TRUE)
P <- matrix(rep(p, n.occasions*Nsuper), ncol = n.occasions, nrow = Nsuper,
            byrow = TRUE)

# Create data frames to store results
js.med.df <- data.frame(model=character(), Nsuper=numeric(),
phi=numeric(),
                        b1=numeric(), b2=numeric(), b3=numeric(),
b4=numeric(),
                         b5=numeric(), b6=numeric(), b7=numeric(),
p=numeric(),
                         f1=numeric(), f2=numeric(), f3=numeric(),
f4=numeric(),
                         f5=numeric(), f6=numeric(), N1=numeric(),
N2=numeric(),
                         N3=numeric(), N4=numeric(), N5=numeric(),
N6=numeric(),
                         N7=numeric(), lambda1=numeric(),
lambda2=numeric(),

lambda3=numeric(),lambda4=numeric(),lambda5=numeric(),
                         lambda6=numeric())

js.sd.df <- data.frame(model=character(), Nsuper=numeric(), phi=numeric(),
                        b1=numeric(), b2=numeric(), b3=numeric(),
b4=numeric(),
                        b5=numeric(), b6=numeric(), b7=numeric(),
p=numeric(),
                        f1=numeric(), f2=numeric(), f3=numeric(),
f4=numeric(),
                        f5=numeric(), f6=numeric(), N1=numeric(),
N2=numeric(),
                        N3=numeric(), N4=numeric(), N5=numeric(),
N6=numeric(),
                        N7=numeric(), lambda1=numeric(), lambda2=numeric(),

lambda3=numeric(),lambda4=numeric(),lambda5=numeric(),
                        lambda6=numeric())

js.025.df <- data.frame(model=character(), Nsuper=numeric(),
phi=numeric(),
                         b1=numeric(), b2=numeric(), b3=numeric(),
b4=numeric(),
                         b5=numeric(), b6=numeric(), b7=numeric(),
p=numeric(),
```

```
                        f1=numeric(), f2=numeric(), f3=numeric(),
f4=numeric(),
                        f5=numeric(), f6=numeric(), N1=numeric(),
N2=numeric(),
                        N3=numeric(), N4=numeric(), N5=numeric(),
N6=numeric(),
                        N7=numeric(), lambda1=numeric(),
lambda2=numeric(),

lambda3=numeric(),lambda4=numeric(),lambda5=numeric(),
                        lambda6=numeric())

js.975.df <- data.frame(model=character(), Nsuper=numeric(),
phi=numeric(),
                        b1=numeric(), b2=numeric(), b3=numeric(),
b4=numeric(),
                        b5=numeric(), b6=numeric(), b7=numeric(),
p=numeric(),
                        f1=numeric(), f2=numeric(), f3=numeric(),
f4=numeric(),
                        f5=numeric(), f6=numeric(), N1=numeric(),
N2=numeric(),
                        N3=numeric(), N4=numeric(), N5=numeric(),
N6=numeric(),
                        N7=numeric(), lambda1=numeric(),
lambda2=numeric(),

lambda3=numeric(),lambda4=numeric(),lambda5=numeric(),
                        lambda6=numeric())

js.mcmc.df <- data.frame(model=character(), rt=numeric(),
min.ess=numeric(),
                         max.ess=numeric(), med.ess=numeric(),
max.Rhat=numeric())

# Simulate data and run models

# define number of iterations
nrep <- 100

js <- function(nrep){

  for (i in 1:nrep){

    # Simulate capture histories
    sim <- simul.js(PHI, P, b, Nsuper)
    CH <- sim$CH
    # Add dummy occasion
    CH.du <- cbind(rep(0, dim(CH)[1]), CH)
    # Augment data
    ns <- 500
    CH.ms <- rbind(CH.du, matrix(0, ncol = dim(CH.du)[2], nrow = ns))
    # Recode CH matrix: 2 = Not captured , 1 = captured
```

```
    CH.ms[CH.ms==0] <- 2
    # collapse to unique capture histories and frequencies
    chsum <- collapse.ch(CH.ms)
    ys <- chsum[[1]]
    FR <- chsum[[2]]
    M <- sum(FR)

    # MCMC settings
    ni <- 20000
    nt <- 3
    nb <- 5000
    nc <- 3

    # Parameters to monitor
    parameters <- c("Nsuper", "mean.phi", "b", "mean.p", "N", "lambda",
                    "f1", "f2")

    # model d ------------------------------------------------------------

    # Data for discrete multistate JS model
    jags.data <- list(y = CH.ms, n.occasions = dim(CH.ms)[2], M =
dim(CH.ms)[1])
    # initial values for s
    sst <- js.multistate.init(CH.du, ns)
    # set initial values
    inits <- function(){list(mean.phi = runif(1, .6, .8),
                             mean.p = runif(1, .4, .6), s = sst)}

    # run model
    js.ms.d <- jags(jags.data, inits, parameters, "js-ms-d.txt", n.chains
= nc,
                    n.thin = nt, n.iter = ni, n.burnin = nb,
parallel=TRUE)

    # extract results

    js.med.df <- rbind(js.med.df, data.frame(model = "d",
                Nsuper = median(js.ms.d$sims.list$Nsuper),
                phi = median(js.ms.d$sims.list$mean.phi),
                b1 = median(js.ms.d$sims.list$b[,1]),
                b2 = median(js.ms.d$sims.list$b[,2]),
                b3 = median(js.ms.d$sims.list$b[,3]),
                b4 = median(js.ms.d$sims.list$b[,4]),
                b5 = median(js.ms.d$sims.list$b[,5]),
                b6 = median(js.ms.d$sims.list$b[,6]),
                b7 = median(js.ms.d$sims.list$b[,7]),
                p = median(js.ms.d$sims.list$mean.p),
                f1 = median(js.ms.d$sims.list$f1[,1]),
                f2 = median(js.ms.d$sims.list$f1[,2]),
                f3 = median(js.ms.d$sims.list$f1[,3]),
                f4 = median(js.ms.d$sims.list$f1[,4]),
                f5 = median(js.ms.d$sims.list$f1[,5]),
                f6 = median(js.ms.d$sims.list$f1[,6]),
```

```
                N1 = median(js.ms.d$sims.list$N[,1]),
                N2 = median(js.ms.d$sims.list$N[,2]),
                N3 = median(js.ms.d$sims.list$N[,3]),
                N4 = median(js.ms.d$sims.list$N[,4]),
                N5 = median(js.ms.d$sims.list$N[,5]),
                N6 = median(js.ms.d$sims.list$N[,6]),
                N7 = median(js.ms.d$sims.list$N[,7]),
                lambda1 = median(js.ms.d$sims.list$lambda[,1]),
                lambda2 = median(js.ms.d$sims.list$lambda[,2]),
                lambda3 = median(js.ms.d$sims.list$lambda[,3]),
                lambda4 = median(js.ms.d$sims.list$lambda[,4]),
                lambda5 = median(js.ms.d$sims.list$lambda[,5]),
                lambda6 = median(js.ms.d$sims.list$lambda[,6])))

    js.sd.df <- rbind(js.sd.df, data.frame(model = "d",
                Nsuper = sd(js.ms.d$sims.list$Nsuper),
                phi = sd(js.ms.d$sims.list$mean.phi),
                b1 = sd(js.ms.d$sims.list$b[,1]),
                b2 = sd(js.ms.d$sims.list$b[,2]),
                b3 = sd(js.ms.d$sims.list$b[,3]),
                b4 = sd(js.ms.d$sims.list$b[,4]),
                b5 = sd(js.ms.d$sims.list$b[,5]),
                b6 = sd(js.ms.d$sims.list$b[,6]),
                b7 = sd(js.ms.d$sims.list$b[,7]),
                p = sd(js.ms.d$sims.list$mean.p),
                f1 = sd(js.ms.d$sims.list$f1[,1]),
                f2 = sd(js.ms.d$sims.list$f1[,2]),
                f3 = sd(js.ms.d$sims.list$f1[,3]),
                f4 = sd(js.ms.d$sims.list$f1[,4]),
                f5 = sd(js.ms.d$sims.list$f1[,5]),
                f6 = sd(js.ms.d$sims.list$f1[,6]),
                N1 = sd(js.ms.d$sims.list$N[,1]),
                N2 = sd(js.ms.d$sims.list$N[,2]),
                N3 = sd(js.ms.d$sims.list$N[,3]),
                N4 = sd(js.ms.d$sims.list$N[,4]),
                N5 = sd(js.ms.d$sims.list$N[,5]),
                N6 = sd(js.ms.d$sims.list$N[,6]),
                N7 = sd(js.ms.d$sims.list$N[,7]),
                lambda1 = sd(js.ms.d$sims.list$lambda[,1]),
                lambda2 = sd(js.ms.d$sims.list$lambda[,2]),
                lambda3 = sd(js.ms.d$sims.list$lambda[,3]),
                lambda4 = sd(js.ms.d$sims.list$lambda[,4]),
                lambda5 = sd(js.ms.d$sims.list$lambda[,5]),
                lambda6 = sd(js.ms.d$sims.list$lambda[,6])))

    js.025.df <- rbind(js.025.df, data.frame(model = "d",
                Nsuper = quantile(js.ms.d$sims.list$Nsuper, probs =
0.025),
                phi = quantile(js.ms.d$sims.list$mean.phi, probs = 0.025),
                b1 = quantile(js.ms.d$sims.list$b[,1], probs = 0.025),
                b2 = quantile(js.ms.d$sims.list$b[,2], probs = 0.025),
                b3 = quantile(js.ms.d$sims.list$b[,3], probs = 0.025),
                b4 = quantile(js.ms.d$sims.list$b[,4], probs = 0.025),
                b5 = quantile(js.ms.d$sims.list$b[,5], probs = 0.025),
```

```
                b6 = quantile(js.ms.d$sims.list$b[,6], probs = 0.025),
                b7 = quantile(js.ms.d$sims.list$b[,7], probs = 0.025),
                p = quantile(js.ms.d$sims.list$mean.p, probs = 0.025),
                f1 = quantile(js.ms.d$sims.list$f1[,1], probs = 0.025),
                f2 = quantile(js.ms.d$sims.list$f1[,2], probs = 0.025),
                f3 = quantile(js.ms.d$sims.list$f1[,3], probs = 0.025),
                f4 = quantile(js.ms.d$sims.list$f1[,4], probs = 0.025),
                f5 = quantile(js.ms.d$sims.list$f1[,5], probs = 0.025),
                f6 = quantile(js.ms.d$sims.list$f1[,6], probs = 0.025),
                N1 = quantile(js.ms.d$sims.list$N[,1], probs = 0.025),
                N2 = quantile(js.ms.d$sims.list$N[,2], probs = 0.025),
                N3 = quantile(js.ms.d$sims.list$N[,3], probs = 0.025),
                N4 = quantile(js.ms.d$sims.list$N[,4], probs = 0.025),
                N5 = quantile(js.ms.d$sims.list$N[,5], probs = 0.025),
                N6 = quantile(js.ms.d$sims.list$N[,6], probs = 0.025),
                N7 = quantile(js.ms.d$sims.list$N[,7], probs = 0.025),
                lambda1 = quantile(js.ms.d$sims.list$lambda[,1], probs =
0.025),
                lambda2 = quantile(js.ms.d$sims.list$lambda[,2], probs =
0.025),
                lambda3 = quantile(js.ms.d$sims.list$lambda[,3], probs =
0.025),
                lambda4 = quantile(js.ms.d$sims.list$lambda[,4], probs =
0.025),
                lambda5 = quantile(js.ms.d$sims.list$lambda[,5], probs =
0.025),
                lambda6 = quantile(js.ms.d$sims.list$lambda[,6], probs =
0.025)))

    js.975.df <- rbind(js.975.df, data.frame(model = "d",
                Nsuper = quantile(js.ms.d$sims.list$Nsuper, probs =
0.975),
                phi = quantile(js.ms.d$sims.list$mean.phi, probs = 0.975),
                b1 = quantile(js.ms.d$sims.list$b[,1], probs = 0.975),
                b2 = quantile(js.ms.d$sims.list$b[,2], probs = 0.975),
                b3 = quantile(js.ms.d$sims.list$b[,3], probs = 0.975),
                b4 = quantile(js.ms.d$sims.list$b[,4], probs = 0.975),
                b5 = quantile(js.ms.d$sims.list$b[,5], probs = 0.975),
                b6 = quantile(js.ms.d$sims.list$b[,6], probs = 0.975),
                b7 = quantile(js.ms.d$sims.list$b[,7], probs = 0.975),
                p = quantile(js.ms.d$sims.list$mean.p, probs = 0.975),
                f1 = quantile(js.ms.d$sims.list$f1[,1], probs = 0.975),
                f2 = quantile(js.ms.d$sims.list$f1[,2], probs = 0.975),
                f3 = quantile(js.ms.d$sims.list$f1[,3], probs = 0.975),
                f4 = quantile(js.ms.d$sims.list$f1[,4], probs = 0.975),
                f5 = quantile(js.ms.d$sims.list$f1[,5], probs = 0.975),
                f6 = quantile(js.ms.d$sims.list$f1[,6], probs = 0.975),
                N1 = quantile(js.ms.d$sims.list$N[,1], probs = 0.975),
                N2 = quantile(js.ms.d$sims.list$N[,2], probs = 0.975),
                N3 = quantile(js.ms.d$sims.list$N[,3], probs = 0.975),
                N4 = quantile(js.ms.d$sims.list$N[,4], probs = 0.975),
                N5 = quantile(js.ms.d$sims.list$N[,5], probs = 0.975),
                N6 = quantile(js.ms.d$sims.list$N[,6], probs = 0.975),
                N7 = quantile(js.ms.d$sims.list$N[,7], probs = 0.975),
```

```
                lambda1 = quantile(js.ms.d$sims.list$lambda[,1], probs =
0.975),
                lambda2 = quantile(js.ms.d$sims.list$lambda[,2], probs =
0.975),
                lambda3 = quantile(js.ms.d$sims.list$lambda[,3], probs =
0.975),
                lambda4 = quantile(js.ms.d$sims.list$lambda[,4], probs =
0.975),
                lambda5 = quantile(js.ms.d$sims.list$lambda[,5], probs =
0.975),
                lambda6 = quantile(js.ms.d$sims.list$lambda[,6], probs =
0.975)))

    js.mcmc.df <- rbind(js.mcmc.df, data.frame(model="d",
                  rt=js.ms.d$mcmc.info$elapsed.mins,
                  min.ess=min(unlist(js.ms.d$n.eff)),
                  max.ess=max(unlist(js.ms.d$n.eff)),
                  med.ess=median(unlist(js.ms.d$n.eff)),
                  max.Rhat=max(unlist(js.ms.d$Rhat))))

    # model m1 -------------------------------------------------------

    # data for marginalized JS model
    jags.data<- list(FR=FR, fr=FR, ys = ys, n.occasions=dim(ys)[2],
MS=dim(ys)[1],
                     M = M)
    # set initial values
    inits <- function(){ list(mean.phi = .45, mean.p = runif(1, .4, .6))}
    # run model
    js.ms.m1 <- jags(jags.data, inits, parameters, "js-ms-m1.txt",
n.chains = nc,
                    n.thin =nt, n.iter = ni, n.burnin = nb, parallel=TRUE)

    # extract results
    js.med.df <- rbind(js.med.df, data.frame(model = "m1",
                 Nsuper = median(js.ms.m1$sims.list$Nsuper),
                 phi = median(js.ms.m1$sims.list$mean.phi),
                 b1 = median(js.ms.m1$sims.list$b[,1]),
                 b2 = median(js.ms.m1$sims.list$b[,2]),
                 b3 = median(js.ms.m1$sims.list$b[,3]),
                 b4 = median(js.ms.m1$sims.list$b[,4]),
                 b5 = median(js.ms.m1$sims.list$b[,5]),
                 b6 = median(js.ms.m1$sims.list$b[,6]),
                 b7 = median(js.ms.m1$sims.list$b[,7]),
                 p = median(js.ms.m1$sims.list$mean.p),
                 f1 = median(js.ms.m1$sims.list$f1[,1]),
                 f2 = median(js.ms.m1$sims.list$f1[,2]),
                 f3 = median(js.ms.m1$sims.list$f1[,3]),
                 f4 = median(js.ms.m1$sims.list$f1[,4]),
                 f5 = median(js.ms.m1$sims.list$f1[,5]),
                 f6 = median(js.ms.m1$sims.list$f1[,6]),
                 N1 = median(js.ms.m1$sims.list$N[,1]),
                 N2 = median(js.ms.m1$sims.list$N[,2]),
```

```
                 N3 = median(js.ms.m1$sims.list$N[,3]),
                 N4 = median(js.ms.m1$sims.list$N[,4]),
                 N5 = median(js.ms.m1$sims.list$N[,5]),
                 N6 = median(js.ms.m1$sims.list$N[,6]),
                 N7 = median(js.ms.m1$sims.list$N[,7]),
                 lambda1 = median(js.ms.m1$sims.list$lambda[,1]),
                 lambda2 = median(js.ms.m1$sims.list$lambda[,2]),
                 lambda3 = median(js.ms.m1$sims.list$lambda[,3]),
                 lambda4 = median(js.ms.m1$sims.list$lambda[,4]),
                 lambda5 = median(js.ms.m1$sims.list$lambda[,5]),
                 lambda6 = median(js.ms.m1$sims.list$lambda[,6])))

    js.sd.df <- rbind(js.sd.df, data.frame(model = "m1",
                 Nsuper = sd(js.ms.m1$sims.list$Nsuper),
                 phi = sd(js.ms.m1$sims.list$mean.phi),
                 b1 = sd(js.ms.m1$sims.list$b[,1]),
                 b2 = sd(js.ms.m1$sims.list$b[,2]),
                 b3 = sd(js.ms.m1$sims.list$b[,3]),
                 b4 = sd(js.ms.m1$sims.list$b[,4]),
                 b5 = sd(js.ms.m1$sims.list$b[,5]),
                 b6 = sd(js.ms.m1$sims.list$b[,6]),
                 b7 = sd(js.ms.m1$sims.list$b[,7]),
                 p = sd(js.ms.m1$sims.list$mean.p),
                 f1 = sd(js.ms.m1$sims.list$f1[,1]),
                 f2 = sd(js.ms.m1$sims.list$f1[,2]),
                 f3 = sd(js.ms.m1$sims.list$f1[,3]),
                 f4 = sd(js.ms.m1$sims.list$f1[,4]),
                 f5 = sd(js.ms.m1$sims.list$f1[,5]),
                 f6 = sd(js.ms.m1$sims.list$f1[,6]),
                 N1 = sd(js.ms.m1$sims.list$N[,1]),
                 N2 = sd(js.ms.m1$sims.list$N[,2]),
                 N3 = sd(js.ms.m1$sims.list$N[,3]),
                 N4 = sd(js.ms.m1$sims.list$N[,4]),
                 N5 = sd(js.ms.m1$sims.list$N[,5]),
                 N6 = sd(js.ms.m1$sims.list$N[,6]),
                 N7 = sd(js.ms.m1$sims.list$N[,7]),
                 lambda1 = sd(js.ms.m1$sims.list$lambda[,1]),
                 lambda2 = sd(js.ms.m1$sims.list$lambda[,2]),
                 lambda3 = sd(js.ms.m1$sims.list$lambda[,3]),
                 lambda4 = sd(js.ms.m1$sims.list$lambda[,4]),
                 lambda5 = sd(js.ms.m1$sims.list$lambda[,5]),
                 lambda6 = sd(js.ms.m1$sims.list$lambda[,6])))

    js.025.df <- rbind(js.025.df, data.frame(model = "m1",
                 Nsuper = quantile(js.ms.m1$sims.list$Nsuper, probs =
0.025),
                 phi = quantile(js.ms.m1$sims.list$mean.phi, probs =
0.025),
                 b1 = quantile(js.ms.m1$sims.list$b[,1], probs = 0.025),
                 b2 = quantile(js.ms.m1$sims.list$b[,2], probs = 0.025),
                 b3 = quantile(js.ms.m1$sims.list$b[,3], probs = 0.025),
                 b4 = quantile(js.ms.m1$sims.list$b[,4], probs = 0.025),
                 b5 = quantile(js.ms.m1$sims.list$b[,5], probs = 0.025),
                 b6 = quantile(js.ms.m1$sims.list$b[,6], probs = 0.025),
```

```
                 b7 = quantile(js.ms.m1$sims.list$b[,7], probs = 0.025),
                 p = quantile(js.ms.m1$sims.list$mean.p, probs = 0.025),
                 f1 = quantile(js.ms.m1$sims.list$f1[,1], probs = 0.025),
                 f2 = quantile(js.ms.m1$sims.list$f1[,2], probs = 0.025),
                 f3 = quantile(js.ms.m1$sims.list$f1[,3], probs = 0.025),
                 f4 = quantile(js.ms.m1$sims.list$f1[,4],probs = 0.025),
                 f5 = quantile(js.ms.m1$sims.list$f1[,5], probs = 0.025),
                 f6 = quantile(js.ms.m1$sims.list$f1[,6], probs = 0.025),
                 N1 = quantile(js.ms.m1$sims.list$N[,1], probs = 0.025),
                 N2 = quantile(js.ms.m1$sims.list$N[,2], probs = 0.025),
                 N3 = quantile(js.ms.m1$sims.list$N[,3], probs = 0.025),
                 N4 = quantile(js.ms.m1$sims.list$N[,4], probs = 0.025),
                 N5 = quantile(js.ms.m1$sims.list$N[,5], probs = 0.025),
                 N6 = quantile(js.ms.m1$sims.list$N[,6], probs = 0.025),
                 N7 = quantile(js.ms.m1$sims.list$N[,7], probs = 0.025),
                 lambda1 = quantile(js.ms.m1$sims.list$lambda[,1], probs =
0.025),
                 lambda2 = quantile(js.ms.m1$sims.list$lambda[,2], probs =
0.025),
                 lambda3 = quantile(js.ms.m1$sims.list$lambda[,3], probs =
0.025),
                 lambda4 = quantile(js.ms.m1$sims.list$lambda[,4], probs =
0.025),
                 lambda5 = quantile(js.ms.m1$sims.list$lambda[,5], probs =
0.025),
                 lambda6 = quantile(js.ms.m1$sims.list$lambda[,6], probs =
0.025)))

    js.975.df <- rbind(js.975.df, data.frame(model = "m1",
                 Nsuper = quantile(js.ms.m1$sims.list$Nsuper, probs =
0.975),
                 phi = quantile(js.ms.m1$sims.list$mean.phi, probs =
0.975),
                 b1 = quantile(js.ms.m1$sims.list$b[,1], probs = 0.975),
                 b2 = quantile(js.ms.m1$sims.list$b[,2], probs = 0.975),
                 b3 = quantile(js.ms.m1$sims.list$b[,3], probs = 0.975),
                 b4 = quantile(js.ms.m1$sims.list$b[,4], probs = 0.975),
                 b5 = quantile(js.ms.m1$sims.list$b[,5], probs = 0.975),
                 b6 = quantile(js.ms.m1$sims.list$b[,6], probs = 0.975),
                 b7 = quantile(js.ms.m1$sims.list$b[,7], probs = 0.975),
                 p = quantile(js.ms.m1$sims.list$mean.p, probs = 0.975),
                 f1 = quantile(js.ms.m1$sims.list$f1[,1], probs = 0.975),
                 f2 = quantile(js.ms.m1$sims.list$f1[,2], probs = 0.975),
                 f3 = quantile(js.ms.m1$sims.list$f1[,3], probs = 0.975),
                 f4 = quantile(js.ms.m1$sims.list$f1[,4],probs = 0.975),
                 f5 = quantile(js.ms.m1$sims.list$f1[,5], probs = 0.975),
                 f6 = quantile(js.ms.m1$sims.list$f1[,6], probs = 0.975),
                 N1 = quantile(js.ms.m1$sims.list$N[,1], probs = 0.975),
                 N2 = quantile(js.ms.m1$sims.list$N[,2], probs = 0.975),
                 N3 = quantile(js.ms.m1$sims.list$N[,3], probs = 0.975),
                 N4 = quantile(js.ms.m1$sims.list$N[,4], probs = 0.975),
                 N5 = quantile(js.ms.m1$sims.list$N[,5], probs = 0.975),
                 N6 = quantile(js.ms.m1$sims.list$N[,6], probs = 0.975),
                 N7 = quantile(js.ms.m1$sims.list$N[,7], probs = 0.975),
```

```
                  lambda1 = quantile(js.ms.m1$sims.list$lambda[,1], probs =
0.975),
                  lambda2 = quantile(js.ms.m1$sims.list$lambda[,2], probs =
0.975),
                  lambda3 = quantile(js.ms.m1$sims.list$lambda[,3], probs =
0.975),
                  lambda4 = quantile(js.ms.m1$sims.list$lambda[,4], probs =
0.975),
                  lambda5 = quantile(js.ms.m1$sims.list$lambda[,5], probs =
0.975),
                  lambda6 = quantile(js.ms.m1$sims.list$lambda[,6], probs =
0.975)))

    js.mcmc.df <- rbind(js.mcmc.df, data.frame(model="m1",
                  rt=js.ms.m1$mcmc.info$elapsed.mins,
                  min.ess=min(unlist(js.ms.m1$n.eff)),
                  max.ess=max(unlist(js.ms.m1$n.eff)),
                  med.ess=median(unlist(js.ms.m1$n.eff)),
                  max.Rhat=max(unlist(js.ms.m1$Rhat))))


    # model m2 -------------------------------------------------------

    all_add<-function(NO_OCC=7){
      # calculates different combinations of arrival and departure timing
      arr<-numeric()
      dep<-numeric()
      t<-1
      for (i in 1:NO_OCC){
        for (j in i:NO_OCC){
          arr[t]<-i
          dep[t]<-j
          t<-t+1
        }}
      return(cbind(arr,dep))}

    MS=dim(ys)[1]

    poss<-all_add()

    first<-function(x){
      #determine timing of first observation
      which(x==1)[1]}

    last<-function(x){
      #determine timing of last observation
      tx<-which(x==1)
      lx<-length(tx)
      tx[lx]
    }

    fd<-apply(ys[-MS,-1],1,first)

```

```
    ld<-apply(ys[-MS,-1],1,last)

    nposs=length(poss[,1])

    poss_obs<-matrix(NA,nrow=nposs,ncol=(MS-1))

    for (i in 1:(MS-1)){
      #for each summarized capture history this loop determines if the
possible
      # underlying dynamics implied by poss are possible
      poss_obs[,i]<-ifelse(poss[,1]<=fd[i]&poss[,2]>=ld[i],1,0)
    }

    Nobs_ind <- sum(FR[-MS])

    lookup <- rep(seq(1,length(FR[-MS]), 1), FR[-MS])

    jags.data<- list(FR=FR, fr=FR, ys = ys, n.occasions=dim(ys)[2],
MS=dim(ys)[1],
                     Nobs_ind = Nobs_ind,Npos_ind = sum(FR) ,
lookup=lookup,
                     nposs=nposs, poss=poss, poss_obs=poss_obs)

    inits <- function(){list(mean.phi = runif(1, .6, .8),
                             mean.p = runif(1, .4, .6),
A=rep(c(fd,NA),FR),
                             L=rep(c(ld,NA),FR), ar=rep(c(fd,1), FR),
                             li=rep(c(ld,(dim(ys)[2]-1)),FR), U=FR[MS])}

    js.ms.m2 <- jags(jags.data, inits, parameters, "js-ms-m2.txt",
n.chains = nc,
                     n.thin =nt, n.iter = ni, n.burnin = nb,
parallel=TRUE)

    # extract results
    js.med.df <- rbind(js.med.df, data.frame(model = "m2",
                  Nsuper = median(js.ms.m2$sims.list$Nsuper),
                  phi = median(js.ms.m2$sims.list$mean.phi),
                  b1 = median(js.ms.m2$sims.list$b[,1]),
                  b2 = median(js.ms.m2$sims.list$b[,2]),
                  b3 = median(js.ms.m2$sims.list$b[,3]),
                  b4 = median(js.ms.m2$sims.list$b[,4]),
                  b5 = median(js.ms.m2$sims.list$b[,5]),
                  b6 = median(js.ms.m2$sims.list$b[,6]),
                  b7 = median(js.ms.m2$sims.list$b[,7]),
                  p = median(js.ms.m2$sims.list$mean.p),
                  f1 = median(js.ms.m2$sims.list$f1[,1]),
                  f2 = median(js.ms.m2$sims.list$f1[,2]),
                  f3 = median(js.ms.m2$sims.list$f1[,3]),
                  f4 = median(js.ms.m2$sims.list$f1[,4]),
                  f5 = median(js.ms.m2$sims.list$f1[,5]),
                  f6 = median(js.ms.m2$sims.list$f1[,6]),
                  N1 = median(js.ms.m2$sims.list$N[,1]),
```

```
                 N2 = median(js.ms.m2$sims.list$N[,2]),
                 N3 = median(js.ms.m2$sims.list$N[,3]),
                 N4 = median(js.ms.m2$sims.list$N[,4]),
                 N5 = median(js.ms.m2$sims.list$N[,5]),
                 N6 = median(js.ms.m2$sims.list$N[,6]),
                 N7 = median(js.ms.m2$sims.list$N[,7]),
                 lambda1 = median(js.ms.m2$sims.list$lambda[,1]),
                 lambda2 = median(js.ms.m2$sims.list$lambda[,2]),
                 lambda3 = median(js.ms.m2$sims.list$lambda[,3]),
                 lambda4 = median(js.ms.m2$sims.list$lambda[,4]),
                 lambda5 = median(js.ms.m2$sims.list$lambda[,5]),
                 lambda6 = median(js.ms.m2$sims.list$lambda[,6])))

    js.sd.df <- rbind(js.sd.df, data.frame(model = "m2",
                Nsuper = sd(js.ms.m2$sims.list$Nsuper),
                phi = sd(js.ms.m2$sims.list$mean.phi),
                b1 = sd(js.ms.m2$sims.list$b[,1]),
                b2 = sd(js.ms.m2$sims.list$b[,2]),
                b3 = sd(js.ms.m2$sims.list$b[,3]),
                b4 = sd(js.ms.m2$sims.list$b[,4]),
                b5 = sd(js.ms.m2$sims.list$b[,5]),
                b6 = sd(js.ms.m2$sims.list$b[,6]),
                b7 = sd(js.ms.m2$sims.list$b[,7]),
                p = sd(js.ms.m2$sims.list$mean.p),
                f1 = sd(js.ms.m2$sims.list$f1[,1]),
                f2 = sd(js.ms.m2$sims.list$f1[,2]),
                f3 = sd(js.ms.m2$sims.list$f1[,3]),
                f4 = sd(js.ms.m2$sims.list$f1[,4]),
                f5 = sd(js.ms.m2$sims.list$f1[,5]),
                f6 = sd(js.ms.m2$sims.list$f1[,6]),
                N1 = sd(js.ms.m2$sims.list$N[,1]),
                N2 = sd(js.ms.m2$sims.list$N[,2]),
                N3 = sd(js.ms.m2$sims.list$N[,3]),
                N4 = sd(js.ms.m2$sims.list$N[,4]),
                N5 = sd(js.ms.m2$sims.list$N[,5]),
                N6 = sd(js.ms.m2$sims.list$N[,6]),
                N7 = sd(js.ms.m2$sims.list$N[,7]),
                lambda1 = sd(js.ms.m2$sims.list$lambda[,1]),
                lambda2 = sd(js.ms.m2$sims.list$lambda[,2]),
                lambda3 = sd(js.ms.m2$sims.list$lambda[,3]),
                lambda4 = sd(js.ms.m2$sims.list$lambda[,4]),
                lambda5 = sd(js.ms.m2$sims.list$lambda[,5]),
                lambda6 = sd(js.ms.m2$sims.list$lambda[,6])))

    js.025.df <- rbind(js.025.df, data.frame(model = "m2",
                 Nsuper = quantile(js.ms.m2$sims.list$Nsuper, probs =
0.025),
                 phi = quantile(js.ms.m2$sims.list$mean.phi, probs =
0.025),
                 b1 = quantile(js.ms.m2$sims.list$b[,1], probs = 0.025),
                 b2 = quantile(js.ms.m2$sims.list$b[,2], probs = 0.025),
                 b3 = quantile(js.ms.m2$sims.list$b[,3], probs = 0.025),
                 b4 = quantile(js.ms.m2$sims.list$b[,4], probs = 0.025),
                 b5 = quantile(js.ms.m2$sims.list$b[,5], probs = 0.025),
```

```
                  b6 = quantile(js.ms.m2$sims.list$b[,6], probs = 0.025),
                  b7 = quantile(js.ms.m2$sims.list$b[,7], probs = 0.025),
                  p = quantile(js.ms.m2$sims.list$mean.p, probs = 0.025),
                  f1 = quantile(js.ms.m2$sims.list$f1[,1], probs = 0.025),
                  f2 = quantile(js.ms.m2$sims.list$f1[,2], probs = 0.025),
                  f3 = quantile(js.ms.m2$sims.list$f1[,3], probs = 0.025),
                  f4 = quantile(js.ms.m2$sims.list$f1[,4], probs = 0.025),
                  f5 = quantile(js.ms.m2$sims.list$f1[,5], probs = 0.025),
                  f6 = quantile(js.ms.m2$sims.list$f1[,6], probs = 0.025),
                  N1 = quantile(js.ms.m2$sims.list$N[,1], probs = 0.025),
                  N2 = quantile(js.ms.m2$sims.list$N[,2], probs = 0.025),
                  N3 = quantile(js.ms.m2$sims.list$N[,3], probs = 0.025),
                  N4 = quantile(js.ms.m2$sims.list$N[,4], probs = 0.025),
                  N5 = quantile(js.ms.m2$sims.list$N[,5], probs = 0.025),
                  N6 = quantile(js.ms.m2$sims.list$N[,6], probs = 0.025),
                  N7 = quantile(js.ms.m2$sims.list$N[,7], probs = 0.025),
                  lambda1 = quantile(js.ms.m2$sims.list$lambda[,1], probs =
0.025),
                  lambda2 = quantile(js.ms.m2$sims.list$lambda[,2], probs =
0.025),
                  lambda3 = quantile(js.ms.m2$sims.list$lambda[,3], probs =
0.025),
                  lambda4 = quantile(js.ms.m2$sims.list$lambda[,4], probs =
0.025),
                  lambda5 = quantile(js.ms.m2$sims.list$lambda[,5], probs =
0.025),
                  lambda6 = quantile(js.ms.m2$sims.list$lambda[,6], probs =
0.025)))

    js.975.df <- rbind(js.975.df, data.frame(model = "m2",
                  Nsuper = quantile(js.ms.m2$sims.list$Nsuper, probs =
0.975),
                  phi = quantile(js.ms.m2$sims.list$mean.phi, probs =
0.975),
                  b1 = quantile(js.ms.m2$sims.list$b[,1], probs = 0.975),
                  b2 = quantile(js.ms.m2$sims.list$b[,2], probs = 0.975),
                  b3 = quantile(js.ms.m2$sims.list$b[,3], probs = 0.975),
                  b4 = quantile(js.ms.m2$sims.list$b[,4], probs = 0.975),
                  b5 = quantile(js.ms.m2$sims.list$b[,5], probs = 0.975),
                  b6 = quantile(js.ms.m2$sims.list$b[,6], probs = 0.975),
                  b7 = quantile(js.ms.m2$sims.list$b[,7], probs = 0.975),
                  p = quantile(js.ms.m2$sims.list$mean.p, probs = 0.975),
                  f1 = quantile(js.ms.m2$sims.list$f1[,1], probs = 0.975),
                  f2 = quantile(js.ms.m2$sims.list$f1[,2], probs = 0.975),
                  f3 = quantile(js.ms.m2$sims.list$f1[,3], probs = 0.975),
                  f4 = quantile(js.ms.m2$sims.list$f1[,4], probs = 0.975),
                  f5 = quantile(js.ms.m2$sims.list$f1[,5], probs = 0.975),
                  f6 = quantile(js.ms.m2$sims.list$f1[,6], probs = 0.975),
                  N1 = quantile(js.ms.m2$sims.list$N[,1], probs = 0.975),
                  N2 = quantile(js.ms.m2$sims.list$N[,2], probs = 0.975),
                  N3 = quantile(js.ms.m2$sims.list$N[,3], probs = 0.975),
                  N4 = quantile(js.ms.m2$sims.list$N[,4], probs = 0.975),
                  N5 = quantile(js.ms.m2$sims.list$N[,5], probs = 0.975),
                  N6 = quantile(js.ms.m2$sims.list$N[,6], probs = 0.975),
```

```
                  N7 = quantile(js.ms.m2$sims.list$N[,7], probs = 0.975),
                  lambda1 = quantile(js.ms.m2$sims.list$lambda[,1], probs =
0.975),
                  lambda2 = quantile(js.ms.m2$sims.list$lambda[,2], probs =
0.975),
                  lambda3 = quantile(js.ms.m2$sims.list$lambda[,3], probs =
0.975),
                  lambda4 = quantile(js.ms.m2$sims.list$lambda[,4], probs =
0.975),
                  lambda5 = quantile(js.ms.m2$sims.list$lambda[,5], probs =
0.975),
                  lambda6 = quantile(js.ms.m2$sims.list$lambda[,6], probs =
0.975)))

    js.mcmc.df <- rbind(js.mcmc.df, data.frame(model="m2",
                   rt=js.ms.m2$mcmc.info$elapsed.mins,
                   min.ess=min(unlist(js.ms.m2$n.eff)),
                   max.ess=max(unlist(js.ms.m2$n.eff)),
                   med.ess=median(unlist(js.ms.m2$n.eff)),
                   max.Rhat=max(unlist(js.ms.m2$Rhat))))


    # model m3 -------------------------------------------------------

    # create index to recover discrete latent states for derived
parameters
    lookup <- rep(seq(1,length(FR), 1), FR)
    jags.data<- list(FR=FR, fr=FR, ys = ys, n.occasions=dim(ys)[2],
MS=dim(ys)[1], M=M,
                     lookup=lookup)
    inits <- function(){list(mean.phi = runif(1, .6, .8), mean.p =
runif(1, .4, .6),
                             U=FR[MS])}
    js.ms.m3 <- jags(jags.data, inits, parameters, "js-ms-m3.txt",
n.chains = nc,
                     n.thin =nt, n.iter = ni, n.burnin = nb,
parallel=TRUE)

    # extract results
    js.med.df <- rbind(js.med.df, data.frame(model = "m3",
                  Nsuper = median(js.ms.m3$sims.list$Nsuper),
                  phi = median(js.ms.m3$sims.list$mean.phi),
                  b1 = median(js.ms.m3$sims.list$b[,1]),
                  b2 = median(js.ms.m3$sims.list$b[,2]),
                  b3 = median(js.ms.m3$sims.list$b[,3]),
                  b4 = median(js.ms.m3$sims.list$b[,4]),
                  b5 = median(js.ms.m3$sims.list$b[,5]),
                  b6 = median(js.ms.m3$sims.list$b[,6]),
                  b7 = median(js.ms.m3$sims.list$b[,7]),
                  p = median(js.ms.m3$sims.list$mean.p),
                  f1 = median(js.ms.m3$sims.list$f1[,1]),
                  f2 = median(js.ms.m3$sims.list$f1[,2]),
                  f3 = median(js.ms.m3$sims.list$f1[,3]),
```

```
                 f4 = median(js.ms.m3$sims.list$f1[,4]),
                 f5 = median(js.ms.m3$sims.list$f1[,5]),
                 f6 = median(js.ms.m3$sims.list$f1[,6]),
                 N1 = median(js.ms.m3$sims.list$N[,1]),
                 N2 = median(js.ms.m3$sims.list$N[,2]),
                 N3 = median(js.ms.m3$sims.list$N[,3]),
                 N4 = median(js.ms.m3$sims.list$N[,4]),
                 N5 = median(js.ms.m3$sims.list$N[,5]),
                 N6 = median(js.ms.m3$sims.list$N[,6]),
                 N7 = median(js.ms.m3$sims.list$N[,7]),
                 lambda1 = median(js.ms.m3$sims.list$lambda[,1]),
                 lambda2 = median(js.ms.m3$sims.list$lambda[,2]),
                 lambda3 = median(js.ms.m3$sims.list$lambda[,3]),
                 lambda4 = median(js.ms.m3$sims.list$lambda[,4]),
                 lambda5 = median(js.ms.m3$sims.list$lambda[,5]),
                 lambda6 = median(js.ms.m3$sims.list$lambda[,6])))

    js.sd.df <- rbind(js.sd.df, data.frame(model = "m3",
                Nsuper = sd(js.ms.m3$sims.list$Nsuper),
                phi = sd(js.ms.m3$sims.list$mean.phi),
                b1 = sd(js.ms.m3$sims.list$b[,1]),
                b2 = sd(js.ms.m3$sims.list$b[,2]),
                b3 = sd(js.ms.m3$sims.list$b[,3]),
                b4 = sd(js.ms.m3$sims.list$b[,4]),
                b5 = sd(js.ms.m3$sims.list$b[,5]),
                b6 = sd(js.ms.m3$sims.list$b[,6]),
                b7 = sd(js.ms.m3$sims.list$b[,7]),
                p = sd(js.ms.m3$sims.list$mean.p),
                f1 = sd(js.ms.m3$sims.list$f1[,1]),
                f2 = sd(js.ms.m3$sims.list$f1[,2]),
                f3 = sd(js.ms.m3$sims.list$f1[,3]),
                f4 = sd(js.ms.m3$sims.list$f1[,4]),
                f5 = sd(js.ms.m3$sims.list$f1[,5]),
                f6 = sd(js.ms.m3$sims.list$f1[,6]),
                N1 = sd(js.ms.m3$sims.list$N[,1]),
                N2 = sd(js.ms.m3$sims.list$N[,2]),
                N3 = sd(js.ms.m3$sims.list$N[,3]),
                N4 = sd(js.ms.m3$sims.list$N[,4]),
                N5 = sd(js.ms.m3$sims.list$N[,5]),
                N6 = sd(js.ms.m3$sims.list$N[,6]),
                N7 = sd(js.ms.m3$sims.list$N[,7]),
                lambda1 = sd(js.ms.m3$sims.list$lambda[,1]),
                lambda2 = sd(js.ms.m3$sims.list$lambda[,2]),
                lambda3 = sd(js.ms.m3$sims.list$lambda[,3]),
                lambda4 = sd(js.ms.m3$sims.list$lambda[,4]),
                lambda5 = sd(js.ms.m3$sims.list$lambda[,5]),
                lambda6 = sd(js.ms.m3$sims.list$lambda[,6])))

    js.025.df <- rbind(js.025.df, data.frame(model = "m3",
                 Nsuper = quantile(js.ms.m3$sims.list$Nsuper, probs =
0.025),
                 phi = quantile(js.ms.m3$sims.list$mean.phi, probs =
0.025),
                 b1 = quantile(js.ms.m3$sims.list$b[,1], probs = 0.025),
```

```
                  b2 = quantile(js.ms.m3$sims.list$b[,2], probs = 0.025),
                  b3 = quantile(js.ms.m3$sims.list$b[,3], probs = 0.025),
                  b4 = quantile(js.ms.m3$sims.list$b[,4], probs = 0.025),
                  b5 = quantile(js.ms.m3$sims.list$b[,5], probs = 0.025),
                  b6 = quantile(js.ms.m3$sims.list$b[,6], probs = 0.025),
                  b7 = quantile(js.ms.m3$sims.list$b[,7], probs = 0.025),
                  p = quantile(js.ms.m3$sims.list$mean.p, probs = 0.025),
                  f1 = quantile(js.ms.m3$sims.list$f1[,1], probs = 0.025),
                  f2 = quantile(js.ms.m3$sims.list$f1[,2], probs = 0.025),
                  f3 = quantile(js.ms.m3$sims.list$f1[,3], probs = 0.025),
                  f4 = quantile(js.ms.m3$sims.list$f1[,4], probs = 0.025),
                  f5 = quantile(js.ms.m3$sims.list$f1[,5], probs = 0.025),
                  f6 = quantile(js.ms.m3$sims.list$f1[,6], probs = 0.025),
                  N1 = quantile(js.ms.m3$sims.list$N[,1], probs = 0.025),
                  N2 = quantile(js.ms.m3$sims.list$N[,2], probs = 0.025),
                  N3 = quantile(js.ms.m3$sims.list$N[,3], probs = 0.025),
                  N4 = quantile(js.ms.m3$sims.list$N[,4], probs = 0.025),
                  N5 = quantile(js.ms.m3$sims.list$N[,5], probs = 0.025),
                  N6 = quantile(js.ms.m3$sims.list$N[,6], probs = 0.025),
                  N7 = quantile(js.ms.m3$sims.list$N[,7], probs = 0.025),
                  lambda1 = quantile(js.ms.m3$sims.list$lambda[,1], probs =
0.025),
                  lambda2 = quantile(js.ms.m3$sims.list$lambda[,2], probs =
0.025),
                  lambda3 = quantile(js.ms.m3$sims.list$lambda[,3], probs =
0.025),
                  lambda4 = quantile(js.ms.m3$sims.list$lambda[,4], probs =
0.025),
                  lambda5 = quantile(js.ms.m3$sims.list$lambda[,5], probs =
0.025),
                  lambda6 = quantile(js.ms.m3$sims.list$lambda[,6], probs =
0.025)))

    js.975.df <- rbind(js.975.df, data.frame(model = "m3",
                  Nsuper = quantile(js.ms.m3$sims.list$Nsuper, probs =
0.975),
                  phi = quantile(js.ms.m3$sims.list$mean.phi, probs =
0.975),
                  b1 = quantile(js.ms.m3$sims.list$b[,1], probs = 0.975),
                  b2 = quantile(js.ms.m3$sims.list$b[,2], probs = 0.975),
                  b3 = quantile(js.ms.m3$sims.list$b[,3], probs = 0.975),
                  b4 = quantile(js.ms.m3$sims.list$b[,4], probs = 0.975),
                  b5 = quantile(js.ms.m3$sims.list$b[,5], probs = 0.975),
                  b6 = quantile(js.ms.m3$sims.list$b[,6], probs = 0.975),
                  b7 = quantile(js.ms.m3$sims.list$b[,7], probs = 0.975),
                  p = quantile(js.ms.m3$sims.list$mean.p, probs = 0.975),
                  f1 = quantile(js.ms.m3$sims.list$f1[,1], probs = 0.975),
                  f2 = quantile(js.ms.m3$sims.list$f1[,2], probs = 0.975),
                  f3 = quantile(js.ms.m3$sims.list$f1[,3], probs = 0.975),
                  f4 = quantile(js.ms.m3$sims.list$f1[,4], probs = 0.975),
                  f5 = quantile(js.ms.m3$sims.list$f1[,5], probs = 0.975),
                  f6 = quantile(js.ms.m3$sims.list$f1[,6], probs = 0.975),
                  N1 = quantile(js.ms.m3$sims.list$N[,1], probs = 0.975),
                  N2 = quantile(js.ms.m3$sims.list$N[,2], probs = 0.975),
```

```
                  N3 = quantile(js.ms.m3$sims.list$N[,3], probs = 0.975),
                  N4 = quantile(js.ms.m3$sims.list$N[,4], probs = 0.975),
                  N5 = quantile(js.ms.m3$sims.list$N[,5], probs = 0.975),
                  N6 = quantile(js.ms.m3$sims.list$N[,6], probs = 0.975),
                  N7 = quantile(js.ms.m3$sims.list$N[,7], probs = 0.975),
                  lambda1 = quantile(js.ms.m3$sims.list$lambda[,1], probs =
0.975),
                  lambda2 = quantile(js.ms.m3$sims.list$lambda[,2], probs =
0.975),
                  lambda3 = quantile(js.ms.m3$sims.list$lambda[,3], probs =
0.975),
                  lambda4 = quantile(js.ms.m3$sims.list$lambda[,4], probs =
0.975),
                  lambda5 = quantile(js.ms.m3$sims.list$lambda[,5], probs =
0.975),
                  lambda6 = quantile(js.ms.m3$sims.list$lambda[,6], probs =
0.975)))

    js.mcmc.df <- rbind(js.mcmc.df, data.frame(model="m3",
                   rt=js.ms.m3$mcmc.info$elapsed.mins,
                   min.ess=min(unlist(js.ms.m3$n.eff)),
                   max.ess=max(unlist(js.ms.m3$n.eff)),
                   med.ess=median(unlist(js.ms.m3$n.eff)),
                   max.Rhat=max(unlist(js.ms.m3$Rhat))))
  }
  return(list(js.med.df, js.sd.df, js.025.df, js.975.df, js.mcmc.df))
}

if (file.exists("js_out_marg.rds")) {
  js.out.marg <- readRDS("js_out_marg.rds")
} else {
  js.out.marg <- js(nrep)  # a time-consuming function
  saveRDS(js.out.marg, "js_out_marg.rds")
}

## extract posterior medians
d.meds <- js.out.marg[[1]][js.out.marg[[1]]$model%in%"d",]
m1.meds <- js.out.marg[[1]][js.out.marg[[1]]$model%in%"m1",]
m2.meds <- js.out.marg[[1]][js.out.marg[[1]]$model%in%"m2",]
m3.meds <- js.out.marg[[1]][js.out.marg[[1]]$model%in%"m3",]

save(d.meds, m1.meds, m2.meds, m3.meds, file = "modelcomparefigS1.RData")


library(kableExtra)
options(knitr.kable.NA = '')

js.out.marg[[5]]$mps100 <-
js.out.marg[[5]]$rt/js.out.marg[[5]]$min.ess*100
js.out.marg[[5]]$mps1000 <-
js.out.marg[[5]]$rt/js.out.marg[[5]]$min.ess*1000

# expected values
```

```
E_N <- c(136, 139, 141, 143, 144, 145, 145)
E_lam <- E_N[2:7]/E_N[1:6]
E_f <- E_lam - .7

# summarize simulations
simsum <- data.frame(model = js.out.marg[[3]]$model,
                     Nsuper.cov = ifelse(js.out.marg[[3]]$Nsuper <= 400 &
                                           js.out.marg[[4]]$Nsuper >= 400,
1, 0),
                     Nsuper.w = js.out.marg[[4]]$Nsuper -
js.out.marg[[3]]$Nsuper,
                     N1.cov = ifelse(js.out.marg[[3]]$N1 <= E_N[1] &
                                     js.out.marg[[4]]$N1 >= E_N[1], 1,
0),
                     N1.w = js.out.marg[[4]]$N1 - js.out.marg[[3]]$N1,
                     N2.cov = ifelse(js.out.marg[[3]]$N2 <= E_N[2] &
                                     js.out.marg[[4]]$N2 >= E_N[2], 1,
0),
                     N2.w = js.out.marg[[4]]$N2 - js.out.marg[[3]]$N2,
                     N3.cov = ifelse(js.out.marg[[3]]$N3 <= E_N[3] &
                                     js.out.marg[[4]]$N3 >= E_N[3], 1,
0),
                     N3.w = js.out.marg[[4]]$N3 - js.out.marg[[3]]$N3,
                     N4.cov = ifelse(js.out.marg[[3]]$N4 <= E_N[4] &
                                     js.out.marg[[4]]$N4 >= E_N[4], 1,
0),
                     N4.w = js.out.marg[[4]]$N4 - js.out.marg[[3]]$N4,
                     N5.cov = ifelse(js.out.marg[[3]]$N5 <= E_N[5] &
                                     js.out.marg[[4]]$N5 >= E_N[5], 1,
0),
                     N5.w = js.out.marg[[4]]$N5 - js.out.marg[[3]]$N5,
                     N6.cov = ifelse(js.out.marg[[3]]$N6 <= E_N[6] &
                                     js.out.marg[[4]]$N6 >= E_N[6], 1,
0),
                     N6.w = js.out.marg[[4]]$N6 - js.out.marg[[3]]$N6,
                     N7.cov = ifelse(js.out.marg[[3]]$N7 <= E_N[7] &
                                     js.out.marg[[4]]$N7 >= E_N[7], 1,
0),
                     N7.w = js.out.marg[[4]]$N7 - js.out.marg[[3]]$N7,
                     lam1.cov = ifelse(js.out.marg[[3]]$lambda1 <=
E_lam[1] &
                                       js.out.marg[[4]]$lambda1 >=
E_lam[1], 1, 0),
                     lam1.w = js.out.marg[[4]]$lambda1 -
js.out.marg[[3]]$lambda1,
                     lam2.cov = ifelse(js.out.marg[[3]]$lambda2 <=
E_lam[2] &
                                       js.out.marg[[4]]$lambda2 >=
E_lam[2], 1, 0),
                     lam2.w = js.out.marg[[4]]$lambda2 -
js.out.marg[[3]]$lambda2,
                     lam3.cov = ifelse(js.out.marg[[3]]$lambda3 <=
E_lam[3] &
                                       js.out.marg[[4]]$lambda3 >=
```

```
E_lam[3], 1, 0),
                     lam3.w = js.out.marg[[4]]$lambda3 -
js.out.marg[[3]]$lambda3,
                     lam4.cov = ifelse(js.out.marg[[3]]$lambda4 <=
E_lam[4] &
                                        js.out.marg[[4]]$lambda4 >=
E_lam[4], 1, 0),
                     lam4.w = js.out.marg[[4]]$lambda4 -
js.out.marg[[3]]$lambda4,
                     lam5.cov = ifelse(js.out.marg[[3]]$lambda5 <=
E_lam[5] &
                                        js.out.marg[[4]]$lambda5 >=
E_lam[5], 1, 0),
                     lam5.w = js.out.marg[[4]]$lambda5 -
js.out.marg[[3]]$lambda5,
                     lam6.cov = ifelse(js.out.marg[[3]]$lambda6 <=
E_lam[6] &
                                        js.out.marg[[4]]$lambda6 >=
E_lam[6], 1, 0),
                     lam6.w = js.out.marg[[4]]$lambda6 -
js.out.marg[[3]]$lambda6,
                     f1.cov = ifelse(js.out.marg[[3]]$f1 <= E_f[1] &
                                     js.out.marg[[4]]$f1 >= E_f[1], 1,
0),
                     f1.w = js.out.marg[[4]]$f1 - js.out.marg[[3]]$f1,
                     f2.cov = ifelse(js.out.marg[[3]]$f2 <= E_f[2] &
                                     js.out.marg[[4]]$f2 >= E_f[2], 1,
0),
                     f2.w = js.out.marg[[4]]$f2 - js.out.marg[[3]]$f2,
                     f3.cov = ifelse(js.out.marg[[3]]$f3 <= E_f[3] &
                                     js.out.marg[[4]]$f3 >= E_f[3], 1,
0),
                     f3.w = js.out.marg[[4]]$f3 - js.out.marg[[3]]$f3,
                     f4.cov = ifelse(js.out.marg[[3]]$f4 <= E_f[4] &
                                     js.out.marg[[4]]$f4 >= E_f[4], 1,
0),
                     f4.w = js.out.marg[[4]]$f4 - js.out.marg[[3]]$f4,
                     f5.cov = ifelse(js.out.marg[[3]]$f5 <= E_f[5] &
                                     js.out.marg[[4]]$f5 >= E_f[5], 1,
0),
                     f5.w = js.out.marg[[4]]$f5 - js.out.marg[[3]]$f5,
                     f6.cov = ifelse(js.out.marg[[3]]$f6 <= E_f[6] &
                                     js.out.marg[[4]]$f6 >= E_f[6], 1,
0),
                     f6.w = js.out.marg[[4]]$f6 - js.out.marg[[3]]$f6)

library(tidyverse)
library(data.table)

x <- simsum %>% group_by(model) %>% summarise(across(.cols=everything(),
                          list(mean = mean, sd = sd), .names =
"{.col}.{.fn}"))

```

```r
xt <- data.table::transpose(x, make.names="model")
xt$parm <- unlist(strsplit(names(x)[2:ncol(x)], "\\.", ""))[seq(1,
length(unlist(strsplit(names(x)[2:ncol(x)], "\\.", "")))), 3)]
xt$var <- rep(c("cov", "cov", "w", "w"), 20)
xt$stat <- rep(c("mean", "sd"), 40)
xt <- xt %>% pivot_wider(id_cols = c(parm, stat), names_from = var,
values_from = c(d, m1, m2, m3))
xt <- data.frame(xt)

xt$d_cov <- sprintf("%.1f", round(xt$d_cov*100, 1))
xt$d_w[xt$parm %in% c("Nsuper", "N1", "N2", "N3", "N4", "N5", "N6", "N7")]
<-
  sprintf("%.1f", round(xt$d_w[xt$parm %in% c("Nsuper", "N1", "N2", "N3",
"N4",
                                                "N5", "N6", "N7")], 1))
xt$d_w[!(xt$parm %in% c("Nsuper", "N1", "N2", "N3", "N4", "N5", "N6",
"N7"))] <-
  sprintf("%.3f", round(as.numeric(xt$d_w[!(xt$parm %in% c("Nsuper", "N1",
                                      "N2", "N3", "N4", "N5", "N6",
"N7"))]), 3))

xt$m1_cov <- sprintf("%.1f", round(xt$m1_cov*100, 1))
xt$m1_w[xt$parm %in% c("Nsuper", "N1", "N2", "N3", "N4", "N5", "N6",
"N7")] <-
  sprintf("%.1f", round(xt$m1_w[xt$parm %in% c("Nsuper", "N1", "N2", "N3",
"N4",
                                                  "N5", "N6", "N7")], 1))
xt$m1_w[!(xt$parm %in% c("Nsuper", "N1", "N2", "N3", "N4", "N5", "N6",
"N7"))] <-
  sprintf("%.3f", round(as.numeric(xt$m1_w[!(xt$parm %in% c("Nsuper",
"N1", "N2",
                                            "N3", "N4", "N5", "N6",
"N7"))]), 3))

xt$m2_cov <- sprintf("%.1f", round(xt$m2_cov*100, 1))
xt$m2_w[xt$parm %in% c("Nsuper", "N1", "N2", "N3", "N4", "N5", "N6",
"N7")] <-
  sprintf("%.1f", round(xt$m2_w[xt$parm %in% c("Nsuper", "N1", "N2", "N3",
"N4",
                                                  "N5", "N6", "N7")], 1))
xt$m2_w[!(xt$parm %in% c("Nsuper", "N1", "N2", "N3", "N4", "N5", "N6",
"N7"))] <-
  sprintf("%.3f", round(as.numeric(xt$m2_w[!(xt$parm %in% c("Nsuper",
"N1", "N2",
                                            "N3", "N4", "N5", "N6",
"N7"))]), 3))

xt$m3_cov <- sprintf("%.1f", round(xt$m3_cov*100, 1))
xt$m3_w[xt$parm %in% c("Nsuper", "N1", "N2", "N3", "N4", "N5", "N6",
"N7")] <-
  sprintf("%.1f", round(xt$m3_w[xt$parm %in% c("Nsuper", "N1", "N2", "N3",
"N4",
                                                  "N5", "N6", "N7")], 1))
```

```
xt$m3_w[!(xt$parm %in% c("Nsuper", "N1", "N2", "N3", "N4", "N5", "N6",
"N7"))] <-
  sprintf("%.3f", round(as.numeric(xt$m3_w[!(xt$parm %in% c("Nsuper",
"N1", "N2",
                                          "N3", "N4", "N5", "N6",
"N7"))]), 3))

xt$parm[seq(2,40,2)] <- NA

xt$d_cov[seq(2,40,2)] <- paste0("(", xt$d_cov[seq(2,40,2)], ")")
xt$d_w[seq(2,40,2)] <- paste0("(", xt$d_w[seq(2,40,2)], ")")
xt$m1_cov[seq(2,40,2)] <- paste0("(", xt$m1_cov[seq(2,40,2)], ")")
xt$m1_w[seq(2,40,2)] <- paste0("(", xt$m1_w[seq(2,40,2)], ")")
xt$m2_cov[seq(2,40,2)] <- paste0("(", xt$m2_cov[seq(2,40,2)], ")")
xt$m2_w[seq(2,40,2)] <- paste0("(", xt$m2_w[seq(2,40,2)], ")")
xt$m3_cov[seq(2,40,2)] <- paste0("(", xt$m3_cov[seq(2,40,2)], ")")
xt$m3_w[seq(2,40,2)] <- paste0("(", xt$m3_w[seq(2,40,2)], ")")

tabS1 <- data.frame(xt[,c(1,3:10)])
tabS1$parm[1] <- "$N_{super}$"
tabS1$parm[3] <- "$N_{1}$"
tabS1$parm[5] <- "$N_{2}$"
tabS1$parm[7] <- "$N_{3}$"
tabS1$parm[9] <- "$N_{4}$"
tabS1$parm[11] <- "$N_{5}$"
tabS1$parm[13] <- "$N_{6}$"
tabS1$parm[15] <- "$N_{7}$"
tabS1$parm[17] <- "$\\lambda_1$"
tabS1$parm[19] <- "$\\lambda_2$"
tabS1$parm[21] <- "$\\lambda_3$"
tabS1$parm[23] <- "$\\lambda_4$"
tabS1$parm[25] <- "$\\lambda_5$"
tabS1$parm[27] <- "$\\lambda_6$"
tabS1$parm[29] <- "$f_1$"
tabS1$parm[31] <- "$f_2$"
tabS1$parm[33] <- "$f_3$"
tabS1$parm[35] <- "$f_4$"
tabS1$parm[37] <- "$f_5$"
tabS1$parm[39] <- "$f_6$"


#-----------------------------------------------------------------------------------------------------------
----
# Create Table S1

kbl(tabS1, format = "pipe", booktabs = TRUE, caption = "Table S1. 95\\%
credible interval coverage and widths [mean (sd)] for discrete (d) and
marginalized (m1, m2, and m3) JS models applied to 100 simulated data sets
that differed with respect to how abundance (N), population change, and
recruitment are estimated.", col.names = c("", "\\% coverage", "Width",
"\\% coverage", "Width", "\\% coverage", "Width", "\\% coverage",
"Width"), align = "c") %>%
  add_header_above(header = c(" ", "d" = 2, "m1" = 2, "m2" = 2, "m3" = 2))
```

```
# note that "add_header_above() not working when knitting document to Word.

#-----------------------------------------------------------------------
---

# Create Figure S1

library(fields)
library(wesanderson)

par(family = "serif", mfrow=c(3,1), mar = c(4,5,0,3))
# in response to review, ad hoc y-axis split, Nsuper shown on right
d.meds$Nsuper <- d.meds$Nsuper - 100
m1.meds$Nsuper <- m1.meds$Nsuper - 100
m2.meds$Nsuper <- m2.meds$Nsuper - 100
m3.meds$Nsuper <- m3.meds$Nsuper - 100

bplot(d.meds[,c(18:24,2)], use.cols = TRUE, outline = FALSE, staplewex=0,
col= wes_palette("Royal1")[1],
      boxwex=.15, lty=1, pos = seq(from=.5, by=2, length=8), xaxt="n",
yaxt="n", ylab = "Abundance (N)",
      xlim=c(0,16), ylim=c(50, 400), las=1, cex.lab=1.25, cex.axis=1)
axis(2, at = c(100, 150, 200), las=1, cex.axis=1.25)
axis(4, at = c(300, 350, 400), labels = c(400, 450, 500), las=1,
cex.axis=1.25)
abline(h=225, lty=2, col="gray")
bplot(m1.meds[,c(18:24,2)], use.cols = TRUE, outline = FALSE, staplewex=0,
col= wes_palette("Royal1")[2],
      boxwex=.15, lty=1, pos = seq(from=.83, by=2, length=8), las=1,
xaxt="n", yaxt="n", add=TRUE)
bplot(m2.meds[,c(18:24,2)], use.cols = TRUE, outline = FALSE, staplewex=0,
col= wes_palette("Royal1")[3],
      boxwex=.15, lty=1, pos = seq(from=1.17, by=2, length=8), las=1,
xaxt="n", yaxt="n", add=TRUE)
bplot(m3.meds[,c(18:24,2)], use.cols = TRUE, outline = FALSE, staplewex=0,
col= wes_palette("Royal1")[4],
      boxwex=.15, lty=1, pos = seq(from=1.5, by=2, length=8), las=1,
xaxt="n", yaxt="n", add=TRUE)
bplot(data.frame(136, 139, 141, 143, 144, 145, 145, 300), xaxt="n",
yaxt="n",
      use.cols = TRUE, boxwex=.8, border= "red", las=1, pos = seq(from=1,
by=2, length=8),
      add=TRUE)
axis(1, at = c(1,3,5,7,9,11,13,15), labels = c(expression(N[1]),
expression(N[2]), expression(N[3]), expression(N[4]), expression(N[5]),
expression(N[6]), expression(N[7]), "Nsuper"), cex.axis=1)
legend("top", legend = c("d", "m1", "m2", "m3" ), fill =
wes_palette("Royal1"),
       bty="n")
title(main = "a", line=-1, adj=0.01, cex.main=1.5)

bplot(d.meds[,25:30], use.cols = TRUE, outline = FALSE, staplewex=0, col=
wes_palette("Royal1")[1],
      boxwex=.15, lty=1, pos = seq(from=.5, by=2, length=6), xaxt="n",
```

```
      ylab = expression(paste("Population change (", lambda, ")"),
sep=""),
      xlim=c(0,12), ylim = c(0.2, 1.7), las=1, cex.lab=1.25, cex.axis=1)
bplot(m1.meds[,25:30], use.cols = TRUE, outline = FALSE, staplewex=0, col=
wes_palette("Royal1")[2],
      boxwex=.15, lty=1, pos = seq(from=.83, by=2, length=6), las=1,
xaxt="n", add=TRUE)
bplot(m2.meds[,25:30], use.cols = TRUE, outline = FALSE, staplewex=0, col=
wes_palette("Royal1")[3],
      boxwex=.15, lty=1, pos = seq(from=1.17, by=2, length=6), las=1,
xaxt="n", add=TRUE)
bplot(m3.meds[,25:30], use.cols = TRUE, outline = FALSE, staplewex=0, col=
wes_palette("Royal1")[4],
      boxwex=.15, lty=1, pos = seq(from=1.5, by=2, length=6), las=1,
xaxt="n", add=TRUE)
bplot(data.frame(E_lam[1], E_lam[2], E_lam[3], E_lam[4], E_lam[5],
E_lam[6]), xaxt="n",
      use.cols = TRUE, boxwex=.8, border= "red", las=1, pos = seq(from=1,
by=2, length=6),
      add=TRUE)
axis(1, at = c(1,3,5,7,9,11), labels = c(expression(italic(lambda[1])),

expression(italic(lambda[2])),

expression(italic(lambda[3])),

expression(italic(lambda[4])),

expression(italic(lambda[5])),

expression(italic(lambda[6]))), cex.axis=1)
title(main="b", line=-1, adj=0.01, cex.main=1.5)


bplot(d.meds[,12:17], use.cols = TRUE, outline = FALSE, staplewex=0, col=
wes_palette("Royal1")[1],
      boxwex=.15, lty=1, pos = seq(from=.5, by=2, length=6), xaxt="n",
      ylab = "Recruitment (f)",
      xlim=c(0,12), ylim = c(0, 1), las=1, cex.lab=1.25, cex.axis=1)
bplot(m1.meds[,12:17], use.cols = TRUE, outline = FALSE, staplewex=0, col=
wes_palette("Royal1")[2],
      boxwex=.15, lty=1, pos = seq(from=.83, by=2, length=6), las=1,
xaxt="n", add=TRUE)
bplot(m2.meds[,12:17], use.cols = TRUE, outline = FALSE, staplewex=0, col=
wes_palette("Royal1")[3],
      boxwex=.15, lty=1, pos = seq(from=1.17, by=2, length=6), las=1,
xaxt="n", add=TRUE)
bplot(m3.meds[,12:17], use.cols = TRUE, outline = FALSE, staplewex=0, col=
wes_palette("Royal1")[4],
      boxwex=.15, lty=1, pos = seq(from=1.5, by=2, length=6), las=1,
xaxt="n", add=TRUE)
bplot(data.frame(E_f[1], E_f[2], E_f[3], E_f[4], E_f[5], E_f[6]),
xaxt="n",
      use.cols = TRUE, boxwex=.8, border= "red", las=1, pos = seq(from=1,
```

```
by=2, length=6),
      add=TRUE)
axis(1, at = c(1,3,5,7,9,11), labels = c(expression(italic(f[1])),
                                         expression(italic(f[2])),
                                         expression(italic(f[3])),
                                         expression(italic(f[4])),
                                         expression(italic(f[5])),
                                         expression(italic(f[6]))),
cex.axis=1)
title(main="c", line=-1, adj=0.01, cex.main=1.5)

#-----------------------------------------------------------------------
---

# Create Figure S2


js.out.marg[[5]]$mps100 <-
js.out.marg[[5]]$rt/js.out.marg[[5]]$min.ess*100
js.out.marg[[5]]$mps1000 <-
js.out.marg[[5]]$rt/js.out.marg[[5]]$min.ess*1000

par(family = "serif", mfrow=c(1,1), mar = c(4,5,0,0)) # family = "CM
Roman",
boxplot(js.out.marg[[5]]$mps1000 ~ js.out.marg[[5]]$model, log = 'y',
xlab= "Model",
        ylab = "Time to 1000 effective samples (min)",
        xaxt= "n", yaxt="n", outline=FALSE, lty=1, staplewex=0)
axis(2, at = c(0, 0.1, 1, 10, 100), las = 1)
axis(1, at = c(1, 2, 3, 4), labels = c("d", "m1", "m2", "m3"), las = 1)
```