Vignette: Simulation and fitting of the count-detection model with the observed-data likelihood when validated recordings are missing at random

Jacob Oram[1] Katharine Banner[1] Christian Stratton[2] Kathryn M. Irvine[3,*]

2024-08-19

Abstract

Our vignette demonstrates the use of software designed to simulate from and fit the count-detection model when validated true species labels are missing at random. Our functions allow the user to specify varying degrees of validation effort (i.e., number of recordings selected for validation) under two possible designs. We provide examples of data simulation, model fitting, MCMC evaluation, and visualization of simulation results under each validation design. Our demonstration here is intended to aid researchers and others to tailor a validation design that provides useful inference while also ensuring that the level of effort meets cost constraints.

[1] Department of Mathematical Sciences, Montana State University, Bozeman, MT, USA
[2] Department of Mathematics and Statistics, Middlebury College, Middlebury, VT, USA
[3] U.S. Geological Survey, Northern Rocky Mountain Science Center, Bozeman, MT, USA

[*] Correspondence: Kathryn M. Irvine <kirvine@usgs.gov>

Contents

1 Introduction

Stationary acoustic detectors provide one of the main sources of data used by the North American Bat Monitoring program to obtain inference about species status and trends (Loeb et al. 2015). However, to obtain reliable inference about bat species occurrence and relative activity, monitoring programs that rely on passive acoustic methods must account for imperfect detection and misclassification by automated software in statistical models Wright et al. (2020). The most realistic model to date is the count-detection framework as proposed by Wright et al. (2020), which relies on expert-validated data to inform the misclassification process model. However, validation of recordings by experts is bottleneck in the acoustic-data workflow that has hampered the adoption of the count-detection framework. In an effort to lower the costs associated with manual verification of machine-generated autoIDs and the use of the count-detection model, we developed the ValidationExplorer, a software tool that streamlines simulation studies so that researchers can determine which validation design (and how many recordings must be validated) to obtain reliable inference.

The user inputs for the ValidationExplorer are

- The type of validation design (i.e., the mechanism by which recordings are selected for validation by experts)
- The possible levels of effort (i.e., given the validation design, how many recordings are selected)
- The assumed occupancy and relative activity parameter values for species of interest.

The output from simulations are numerical summaries of the number of recordings validated as well as numerical and visual summaries of posterior means, 95% posterior intervals, estimation error and coverage for each parameter in the model.

In this document, we demonstrate the use of the ValidationExplorer software, including the simulation of synthetic data under the count-detection model framework, the subsequent model-fitting process when accounting for the validation design, and the visualization of simulation results to aid in the decision process. Our software provides the option to specify two validation designs: a stratified-by-species validation design and a fixed-effort design, as described by Oram et al., (in press). We provide detailed step-by-step instructions for using a stratified-by-species design to validate acoustic recordings that have already been classified by automated software algorithms. We also provide a streamlined example of simulations under the fixed-effort design, as well as a mock decision-making process given a feasible number of recordings. All simulation assume an observation-level form of the count-detection model, described in the following subsection.

1.1 Model specification

We assume that the inferential goal is to estimate the occurrence probability and relative activity levels for all species in a multi-species assemblage while accounting for errors in the detection and classification process. The model formulation assumes a latent occupancy indicator, denoted as Z_{ik}, so that

$$Z_{ik} = \begin{cases} 1 \text{ if species } k \text{ is present at site } i \\ 0 \text{ otherwise} \end{cases} .$$

The occupancy indicator is modeled as a Bernoulli random variable with occurrence probability ψ_k

$$Z_{ik} \sim \text{Bernoulli}(\psi_k).$$

In the ValidationExplorer, we assume that each species has a occurrence probability ψ_k that is constant across sites. In the future, we intend to extend the model to accommodate covariates in a generalized linear model framework so that species occurrence may change based on relevant covariates.

Next, let $\mathbf{Z}_i = (Z_{i1}, Z_{i2}, \ldots, Z_{iK})'$ denote the vector of species occurrence indicators at location i. Further, Let $j = 1, 2, \ldots, J$ index a detector night. At locations where species k is present (i.e., $Z_{ik} = 1$), we model the number of recordings from that species on visit j to site i as a Poisson random variable with rate λ_k, representing the relative activity of species k:

$$Y_{ijk} | Z_{ik} = 1 \sim \text{Poisson}(\lambda_{ijk}).$$

We assume independence of Y_{ijk} across species. Furthermore, we assume that each species has a single relative activity paramter that is constant across detector nights. In the presence of an imperfect classification algorithm, we do not directly observe Y_{ijk}. However, we do observe the total number of calls from the site-night, which is denoted as $Y_{ij\cdot} = \sum_{k=1}^{K} Y_{ijk}$. We assume independence across visits j and model $Y_{ij\cdot}$ as a mixture distribution with components defined by the species occurrence indicators $\mathbf{Z}_i$:

$$Y_{ij\cdot} | \mathbf{Z}_i \sim \begin{cases} 0 \text{ with probability } \prod_k (1 - \psi_{ik}) \\ \text{Poisson}\left(\mathbf{Z}_i' \boldsymbol{\lambda}\right) \text{ with probability } (1 - \prod_k (1 - \psi_{ik})) \end{cases} \tag{1}$$

In Equation 1, $\boldsymbol{\lambda}$ denotes a $K \times 1$ vector where the k^{th} entry contains the relative activity parameter for species k.

At locations where observations were made (i.e., at least one species was present and recorded so that $Y_{ij\cdot} > 0$), let $l_{ij} = 1, 2, \ldots, Y_{ij\cdot}$ index an individual observation within a site-night (i, j). Further let, $\mathbf{T}_{l_{ij}}$ and $\mathbf{A}_{l_{ij}}$ denote the $K \times 1$ vectors indicating the true and autoID species labels, respectively. We sometimes write $T_{l_{ij}} = k$ to denote $\mathbf{T}_{l_{ij}} = (T_{l_{ij}1}, T_{l_{ij}2}, \ldots, T_{l_{ij}k}, \ldots, T_{l_{ij}K})' = (0, 0, \ldots, 0, 1, 0, \ldots, 0)'$, where the only non-zero entry is the k^{th} element. After manually validating a recording, the true species label $\mathbf{T}_{l_{ij}}$ is known. We assume that human-validated true species labels are always correct and refer to the subset of validated recordings with known true labels as the validation set.

Due to properties of independent Poisson random variables, and conditional on the observed total count of recordings $Y_{ij\cdot} = y_{ij\cdot} > 0$, the number of recordings due to true species k is a Multinomial random variable with probability vector $\mathbf{p}_i$ (defined below). Assuming independence among recordings, we model the true species label of an individual recording, $\mathbf{T}_{l_{ij}}$ as a multinomial random variable with the same probability vector $\mathbf{p}_i$:

$$\begin{aligned} \mathbf{T}_{l_{ij}} | \mathbf{Z}_i, \boldsymbol{\lambda} \sim & \text{Multinomial}(1, \mathbf{p}_i), \;\; l_{ij} = 1, 2, \ldots, y_{ij\cdot} \\ \mathbf{p}_i = & (p_{i1}, p_{i2}, \ldots p_{iK}) \\ p_{ik} = & \frac{Z_{ik}\lambda_k}{\sum_{m=1}^{K} Z_{im}\lambda_m}. \end{aligned} \tag{2}$$

The k^{th} entry in $\mathbf{p}$, p_{ik}, corresponds to the probability that a recording observed at location i is attributable to species k.

Conditional on the true species label, we model the classification process through a classification matrix Θ, where entry $\{k, k^{\star}\} = \theta_{kk^{\star}}$ denotes the probability that a recording from true species k is assigned autoID label $k^{\star}$. Note that diagonal entries in this classification matrix correspond to the probability of correct classification and the rows of Θ sum to 1: $\sum_m \theta_{km} = 1$. Given the true species label for recording l_{ij}, the autoID $\mathbf{A}_{l_{ij}}$ is modeled as a multinomial random variable with the probability vector determined by row of Θ corresponding to the true species that generated the recording:

$$\mathbf{A}_{l_{ij}} | \mathbf{T}_{l_{ij}}, \Theta \sim \text{Multinomial}(1, \mathbf{T}'_{l_{ij}}\Theta). \tag{3}$$

As with $\mathbf{T}_{l_{ij}}$, we sometimes use $A_{l_{ij}} = m$ as a notational shorthand for $\mathbf{A}_{l_{ij}} = (A_{l_{ij}1}, \ldots, A_{l_{ij}m-1}, A_{l_{ij}m}, A_{l_{ij}m+1}, \ldots, A_{l_{ij}K}) = (0, \ldots, 0, 1, 0, \ldots, 0)'$, where the 1 is in the m^{th} entry.

To complete the model, we must specify prior distributions on the classification matrix elements, Θ, the relative activity rates $\boldsymbol{\lambda}$ and the occurrence probabilities $\boldsymbol{\psi}$.

We adopt the priors used by Stratton et al. (2022): independent $\text{Beta}(1,1)$ priors for each ψ_k, independent half-normal priors $N_+(0, \sigma = 100)$ for each λ_k, and $\text{Dirichlet}(\boldsymbol{\alpha})$ priors on each row of Θ. We set the concentration vector $\boldsymbol{\alpha} = 1/K \times \mathbf{1}$, the reference distance prior recommended by Stratton et al. (2022).

The ValidationExplorer automates the specification and model-fitting process for this framework using the NIMBLE package Valpine et al. (2017).

2 Conducting your own simulation study: stratified-by-species designs

The goal of the ValidationExplorer software is to provide insight about how validation design (and effort) may influence inference. Assuming that the data arise according to the model in Section @ref{model}, will a particular level of effort under a specific validation design be expected to yield enough information to reliably estimate species occurrence and relative activity?

For our first example simulation study, we assume a stratified-by-species validation design. Briefly, this design takes a stratified random sample with unequal probabilities within each stratum, which is formed by the AutoID labels. The motivation for this design is to allow monitoring programs to strategically tailor the available validation effort to prioritize species of greater conservation concern. The purpose of this example is to demonstrate the mechanics of using the functions in greater detail. For a mock simulation study that includes an example decision-making process, see Section **Y**.

2.1 Step 1: Installing and loading required packages

After cloning this repo, the next step is loading the necessary packages in R. To simulate, fit and visualize models used with a stratified-by-species validation design using our code, the following packages are necessary:

- `tidyverse`

- `nimble`
- `coda`
- `rstan`
- `parallel`
- `here`
- `viridis`
- `bayesplot` (optional)

If you do not have one or more of these packages installed, run the following, with the name of the missing package in place of `your_package_name_here`:

```
install.packages("your_package_name_here")
```

After installing the necessary packages, load these libraries by calling

```
library(tidyverse)
library(nimble)
library(coda)
library(rstan)
library(parallel)
library(here)
```

2.2 Step 2: Set up your working directory

In the course of the simulation study, several objects are saved:

- simulated datasets (optional)
- simulated datasets after validation (optional)
- model fits (optional)
- individual summaries for one dataset/validation scenario combination (optional)
- overall summaries for each validation scenario (always saved)

Functions that save any part of the simulation require a `directory` argument be specified. If you choose to save model fits or individual summaries, our simulation functions expect your working directory to contain the folders

- `PathToYourWorkingDirectory/ThetaID/fits`

- `PathToYourWorkingDirectory/ThetaID/individual_summaries`

Above, `PathToYourWorkingDirectory` is replaced with the file path to your working directory (e.g. `~/Documents` for the local Documents folder on Mac) and `ID` is replaced with the classifier scenario ID. Visually, this file structure should appear as follows:

- YourWorkingDirectory
  - ThetaID
    - fits
    - individual_summaries

See the Testing folder in this repo for an example (ignore the blank placeHold.txt files, which are simply there to retain the empty directory structure). Here, we use numbers for the classifier scenarios. The first scenario has `ID=1`, giving the paths to the necessary directories `your/working/directory/Testing/Theta1/fits` and `your/working/directory/Testing/Theta1/individual_summaries`.

2.3 Step 3: Simulate data

With the required folder structure set up, we can now simulate data using stratified-by-species validation. This is accomplished using the `simulate_validatedData` function. Load this function by running

```r
source("../Data Simulation/simulate_validatedData.R")
```

Next, determine the species assemblage to simulate, and the number of sites and visits. We start by assigning the occurrence and relative activity parameters for the three species, then define the number of sites and visits.

```r
psi <- c(0.3, 0.6, 0.9)
lambda <- c(11, 2, 4)

# Define sites and visits
nspecies <- length(psi)
nsites <- 100
nvisits <- 4
```

The data simulation function also requires that we define the classifier and validation scenarios to be used in the simulations. Here, the classifier is encoded as a matrix with rows corresponding to the true species that generated the call, and columns corresponding to the assigned autoID label. Thus the i, j^{th} entry of the matrix (denoted θ_{ij}) is the probability that species i is classified as species j. In `test_theta1` below, the probability that species 2 is classified as species 1 is 0.1, and the probability that species 3 is correctly classified as species 3 is 0.95.

To define a classifier, you can manually enter a matrix:

```r
test_theta1 <- matrix(c(0.9, 0.05, 0.05,
                        0.1, 0.85, 0.05,
                        0.02, 0.03, 0.95), byrow = TRUE, nrow = 3)

test_theta1
```

```
##      [,1] [,2] [,3]
## [1,] 0.90 0.05 0.05
## [2,] 0.10 0.85 0.05
## [3,] 0.02 0.03 0.95
```

This can also be accomplished by using the `rdirch` function from NIMBLE:

```r
test_theta2 <- t(apply(18*diag(nspecies) + 2, 1, function(x) nimble::rdirch(alpha = x)))
test_theta2
```

```
##            [,1]       [,2]      [,3]
## [1,] 0.69033284 0.16668447 0.1429827
## [2,] 0.04456979 0.81123612 0.1441941
## [3,] 0.11857494 0.04115405 0.8402710
```

However you choose to generate the Θ matrix, make sure that the rows sum to 1:

```r
rowSums(test_theta1)
```

```
## [1] 1 1 1
```

```r
rowSums(test_theta2)
```

```
## [1] 1 1 1
```

The next input that needs to be defined for the simulation is the validation efforts for each species label. These are to be stored in a dataframe with each row corresponding to a validation design you would like to investigate. In a simple scenario with the three species above and two levels of effort for each, we could generate an appropriate dataframe by running `expand.grid`. This yields 8 distinct validation scenarios:

```r
val_scenarios <- expand.grid(spp1 = c(.75, .5), spp2 = c(.25, .5), spp3 = c(.25, .75))
val_scenarios
```

```
##   spp1 spp2 spp3
## 1 0.75 0.25 0.25
## 2 0.50 0.25 0.25
## 3 0.75 0.50 0.25
## 4 0.50 0.50 0.25
## 5 0.75 0.25 0.75
## 6 0.50 0.25 0.75
## 7 0.75 0.50 0.75
## 8 0.50 0.50 0.75
```

```r
nrow(val_scenarios)
```

```
## [1] 8
```

With the appropriate inputs defined, we can simulate data. Note that in this example we opt to save both the simulated datasets with all true species labels retained (`save_datasets = TRUE`), as well as the simulated datasets with all true species labels masked except for those that were validated according to the validation scenario (`save_masked_datasets = TRUE`). These datasets are saved in the Testing folder of the current working directory. Note that if you specify the directory using `here::here()`, as we have, the subfolder must have a slash in front of it (e.g., `"/Testing"`). Note that every dataset under every validation set is saved separately, meaning that there will be 2× `n_datasets` + `n_datasets` × `nrow(scenarios)` objects saved in your directory!

```r
fake_data <- simulate_validatedData(
  n_datasets = 5,
  validation_design = "BySpecies",
  scenarios = val_scenarios,
  nsites = nsites,
  nvisits = nvisits,
  nspecies = nspecies,
  psi = psi,
  lambda = lambda,
  theta = test_theta2,
  save_datasets = TRUE,
  save_masked_datasets = TRUE,
  directory = paste0(here::here("Testing"))
)
```

To see what is available, we can investigate `fake_data`. The output is a list, containing three objects:

- `full_datasets`: A list of length `n_datasets` with unmasked datasets (i.e., full validation of all recordings). If `save_datasets = TRUE`, then these will be saved individually in `directory` as dataset_n.rds, where n is the dataset number.
- `zeros`: A list of length `n_datasets` containing all of the site-visits where no recordings of a certain classification were observed. For example, if, in dataset 10, there were no calls from species 1 that were classified as 3 on visit 4 to site 156, then the 10th entry of this list would contain a dataset with a row corresponding to site = 156, visit = 4, true_spp = 1, id_spp = 3, with count = 0. These zeros are necessary for housekeeping in the model-fitting process. If `save_datasets = TRUE`, the zeros for each dataset will be saved in `directory` individually as site_visits_without_calls_in_dataset_n.rds, where n is the dataset number.
- `masked_dfs`: A nested list containing each dataset masked under each scenario. For example, `masked_dfs[[9]][[27]]` contains dataset 27, assuming validation scenario 9. If `save_masked_datasets = TRUE`, then each dataset/scenario combination is saved individually in `directory` as dataset_n_masked_under_scenario_s.rds, where n is the dataset number and s is the scenario number.

Examples of each are given below:

```
full_dfs <- fake_data$full_datasets
head(full_dfs[[1]])
```

```
## # A tibble: 6 x 10
## # Groups:   site, visit [1]
##    site visit true_spp id_spp lambda   psi theta     z count    Y.
##   <int> <int>    <int>  <int>  <dbl> <dbl> <dbl> <int> <int> <int>
## 1     1     1        2      2      2   0.6 0.811     1     1     9
## 2     1     1        3      3      4   0.9 0.840     1     8     9
## 3     1     1        3      3      4   0.9 0.840     1     8     9
## 4     1     1        3      3      4   0.9 0.840     1     8     9
## 5     1     1        3      3      4   0.9 0.840     1     8     9
## 6     1     1        3      3      4   0.9 0.840     1     8     9
```

```
site_visits_without_calls <- fake_data$zeros
head(site_visits_without_calls[[1]])
```

```
## # A tibble: 6 x 10
## # Groups:   site, visit [1]
##    site visit true_spp id_spp lambda   psi  theta     z count    Y.
##   <int> <int>    <int>  <int>  <dbl> <dbl>  <dbl> <int> <int> <int>
## 1     1     1        1      1     11   0.3 0.690      0     0     9
## 2     1     1        2      1      2   0.6 0.0446     1     0     9
## 3      1     1         3      1      4   0.9 0.119      1     0     9
## 4      1     1         1      2     11   0.3 0.167      0     0     9
## 5      1     1         3      2      4   0.9 0.0412     1     0     9
## 6      1     1         1      3     11   0.3 0.143      0     0     9
```

```r
masked_dfs <- fake_data$masked_dfs

# View dataset 3 with scenario 7 validation effort.
head(masked_dfs[[7]][[3]])
```

```
## # A tibble: 6 x 12
## # Groups:   site, visit [1]
##    site visit true_spp id_spp lambda   psi theta     z count    Y.  call
##   <int> <int>    <int>  <int>  <dbl> <dbl> <dbl> <int> <int> <int> <int>
## 1     1     1        3      3      4   0.9 0.840     1     9     9     1
## 2     1     1        3      3      4   0.9 0.840     1     9     9     2
## 3     1     1        3      3      4   0.9 0.840     1     9     9     3
## 4     1     1       NA      3      4   0.9 0.840     1     9     9     4
## 5     1     1        3      3      4   0.9 0.840     1     9     9     5
## 6     1     1        3      3      4   0.9 0.840     1     9     9     6
## # i 1 more variable: scenario <int>
```

For most simulations, it will be useful to summarize the number of recordings that are validated under a given validation design and scenario. This can be accomplished using the `summarize_n_validated` function, which outputs a vector containing the average number of recordings validated in a dataset under each scenario. Note that in our example, there are 8 scenarios (each contained in a row of the `val_scenarios` dataframe), meaning that the output is of length 8:

```r
source("../Data Simulation/summarize_n_validated.R")
summarize_n_validated(data_list = fake_data$masked_dfs)
```

```
## [1] 1361.6 1082.8 1535.8 1257.0 2062.0 1783.2 2236.2 1957.4
```

2.4 Step 4: Fit the model

With the simulated data stored in the global environment, we can now fit models to the simulated datasets using NIMBLE via the wrapper function `run_sims`. This function requires specification of the following arguments:

- `data_list`: The list of masked datasets output from `simulate_validatedData`.
- `zeros_list`: The list of site-visit-true-autoID combinations that were never observed. This is the `zeros` object output from `simulate_validatedData`.

- theta_scenario_id: An ID to show which classifier the scenario is under. This must match the name of the directory .../ThetaID if you want to save model fits and summaries.
- parallel: A logical indicating whether MCMC sampling should be fit in parallel (default setting is TRUE). If you have many datasets and many scenarios, we recommend this setting.
- initialize_lambda_near_naive_val: A logical indicating whether to initialize chains for λ_k near the naive average count of recordings from each species.
- n_iter: The number of iterations for each chain in the MCMC. Default value is 2000.
- nburn: The number of warmup iterations before MCMC draws are retained. By default, half of n_iter draws are discarded as warmup.
- thin: Thinning of the MCMC chains.
- save_fits: A logical denoting whether or not to save the draws from individual fitted models. If TRUE, you must have the file structure described in Step 2. Fits will be saved as RDS objects that can be read in later. Default value is FALSE.

- save_individual_summaries_list: A logical indicating whether to save individual summary lists that are output after each validation scenario. Defalut value is FALSE
- directory: where saved fits and summaries should be saved. Required if save_fits = TRUE or save_individual_summaries_list = TRUE. Default value is the current working directory given by here::here().

An example of using this function with the simulated data from step 3 is given below:

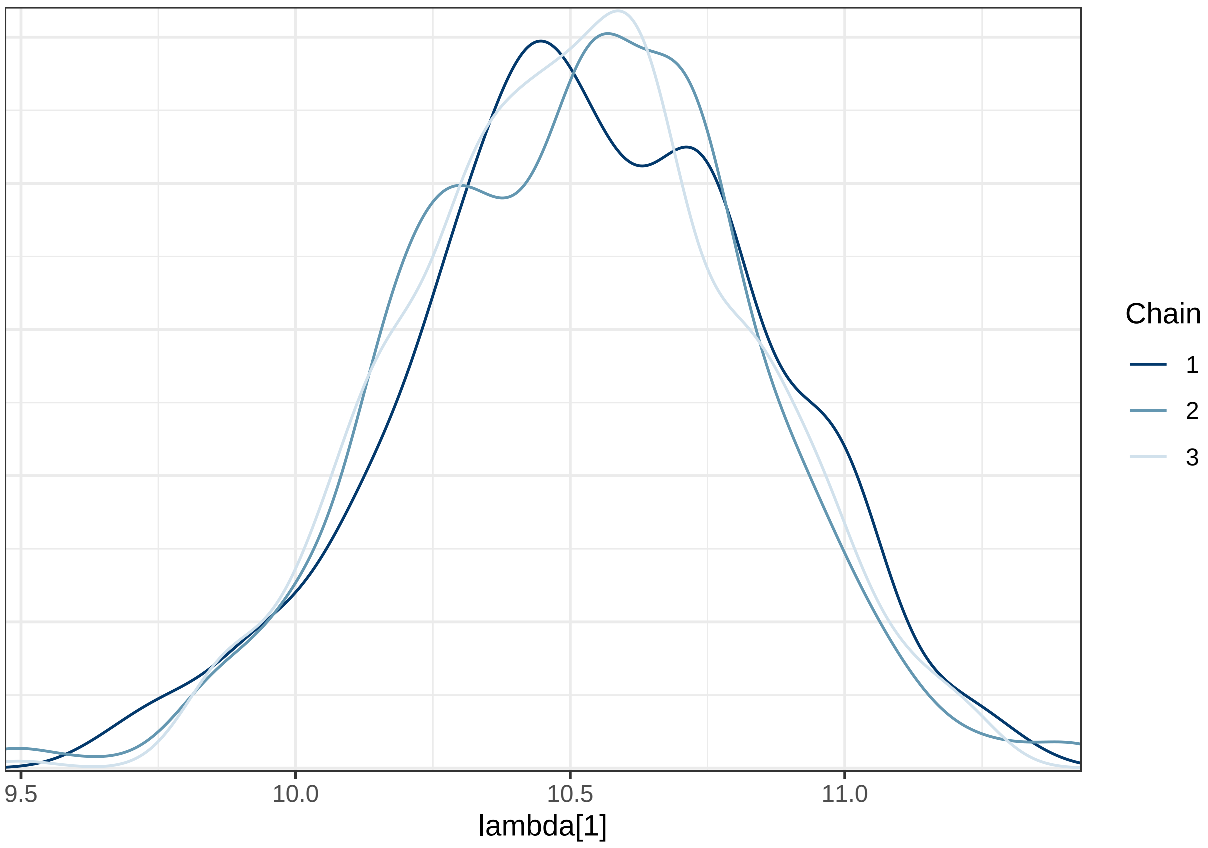
```r
source("../Model Fitting & Simulation/run_sims.R")
# takes about 40 minutes with 10k draws for 5 datasets
sims_output <- run_sims(
         data_list = fake_data$masked_dfs,
         zeros_list = fake_data$zeros,
         DGVs = list(lambda = lambda, psi = psi, theta = test_theta2),
         theta_scenario_id = 1, parallel = TRUE,
         niter = 2000, thin = 1,
         save_fits = TRUE,
         save_individual_summaries_list = FALSE,
         directory = here("Testing"))
```

2.5 Step 5: Assess MCMC convergence

If you selected save_fits = TRUE in the run_sims function, then individual model fits will be available in your/directory/fits. You can use these, together with the Bayesplot package (run `install.packages("bayesplot")` and then `library(bayesplot)` if you do not have this package installed), to visualize model fits through a wide variety of plots. For example, if you would like to see a density plot for species 1's relative activity level in the first dataset under the first validation scenario, you would read in the fit and visualize using `mcmc_dens_overlay`:

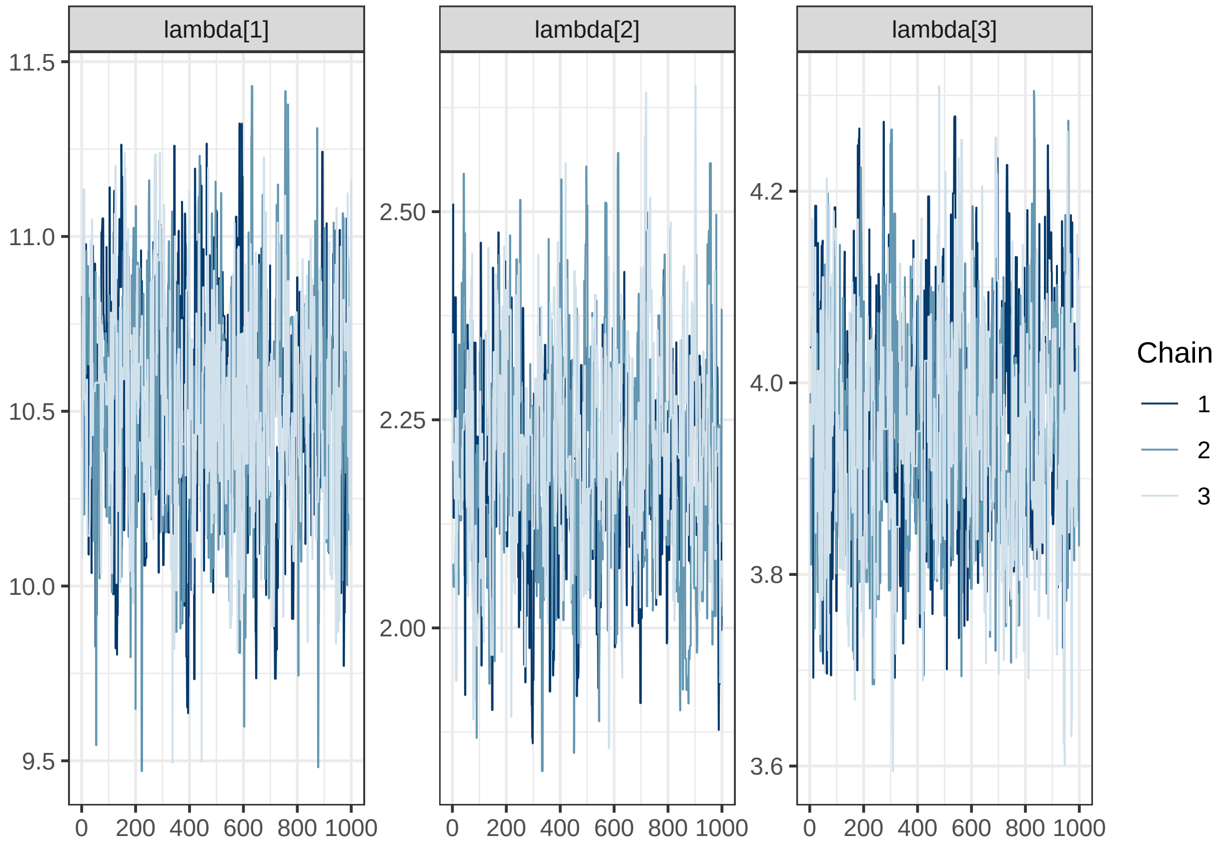
```r
# read in fit object
fit_1_1 <- readRDS("../Testing/Theta1/fits/fit_1_1.rds")

# visualize using bayesplot
bayesplot::mcmc_dens_overlay(fit_1_1, pars = "lambda[1]")
```



To see a traceplot for all *lambda* parameters, run the following:

```r
bayesplot::mcmc_trace(fit_1_1, regex_pars = "lambda")
```



The Bayesplot package has many other visualizations that are available. See their website for more examples and details (http://mc-stan.org/bayesplot/).

The density plots show substantial overlap of the three bell-shaped chains, indicating that the MCMC algorithm has converged. This is further supported by visual inspection of traceplots, which show good mixing. In a simulation study, visual inspection of all traceplots for all parameters in each fitted model is infeasible. Instead, we recommend checking summary tables to make sure that a dataset is included only if all parameters in the model fit to that dataset have Gelman-Rubin statistics $\hat{R} \leq 1.1$. This can be achieved using functions from the dplyr package, where we can see that all parameters were below 1.1, in all five datasets under scenarios 1-4 (which are the only ones considered here).

```
sims_output %>%
  group_by(theta_scenario, scenario, dataset) %>%
  mutate(all_pars_converged = ifelse(unique(converge) == 1, 1, 0)) %>%
  filter(all_pars_converged == 1) %>%
  ungroup () %>%
  select(theta_scenario, scenario, dataset) %>%
  distinct()
```

```
## # A tibble: 20 x 3
##    theta_scenario scenario dataset
##              <dbl>    <int>   <int>
##  1               1        1       1
##  2               1        1       2
##  3               1        1       3
##  4               1        1       4
##  5               1        1       5
##  6               1        2       1
##  7               1        2       2
##  8               1        2       3
##  9               1        2       4
## 10               1        2       5
## 11               1        3       1
## 12               1        3       2
## 13               1        3       3
## 14               1        3       4
## 15               1        3       5
## 16               1        4       1
## 17               1        4       2
## 18               1        4       3
## 19               1        4       4
## 20               1        4       5
```

Note that the visualization functions described in the next section automatically filter to only include model results if the MCMC algorithm shows evidence of convergence.

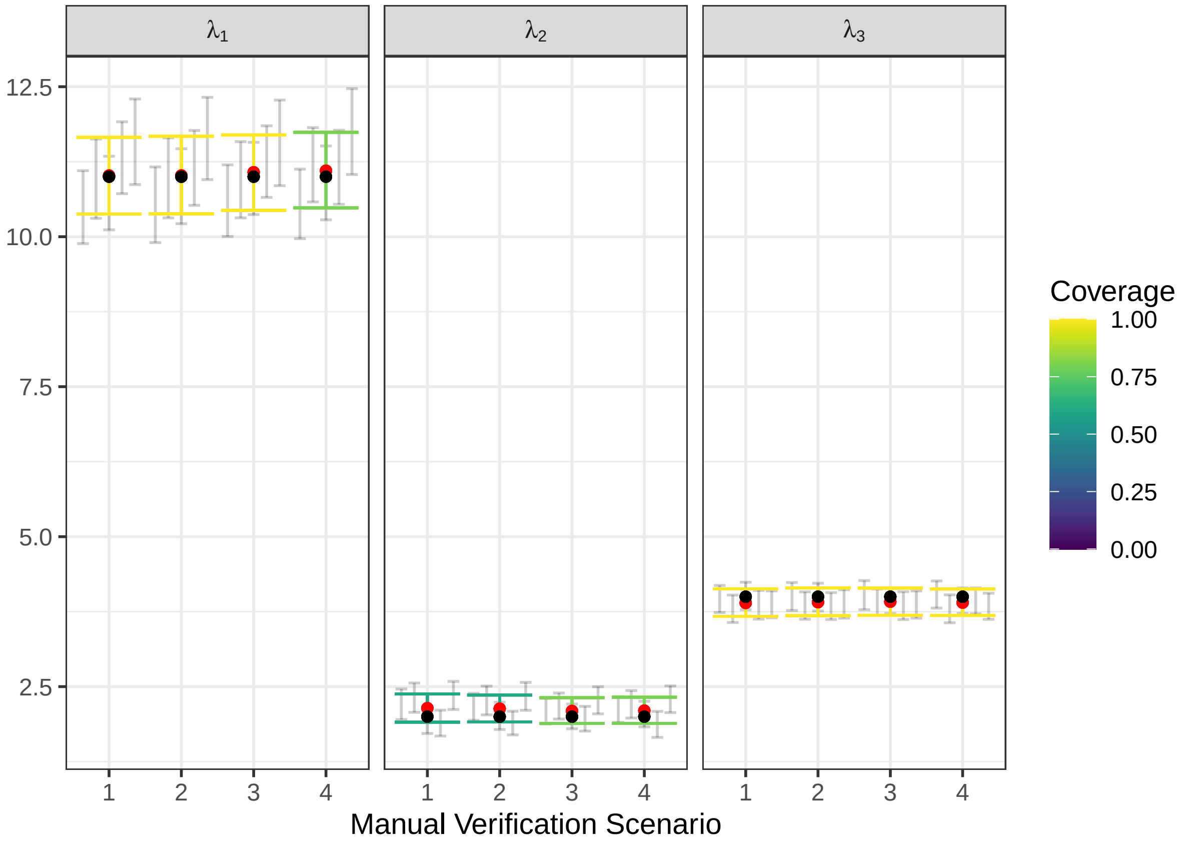
2.6 Step 6: Visualize simulations

Once the simulation study is complete, you can visualize the results using two functions, `visualize_parameter_group` and `visualize_single_parameter`. These functions ensure that only converged models are included in the visualization. `visualize_parameter_group` is useful for examining an entire set of parameters, such as all relative activity parameters. The set of expected inputs for `visualize_parameter_group` are:

- `sim_summary`: A dataframe in the format of the summaries output by `run_sims`. Column names must match those of the `run_sims` output.
- `pars`: The name of the parameter "group" to be visualized (e.g, "psi", "lambda" or "theta").
- `theta_scenario`: The Θ classifier ID.
- `scenarios`: Which scenarios to visualize?
- `convergence_threshold`: What value should $\hat{R}$ be below to be considered "converged"? Default value is 1.1. This value matters because only model fits where all parameter values are below the `convergence_threshold` are used for visualization.

We can visualize the inference for the relative activity parameters in the first three scenarios in our simulation study above by running the code below. Note that we have set `convergence_threshold` to be unrealistically high for illustration purposes. Typically, the default value of 1.1 is as high of an $\hat{R}$ value as possible for the MCMC chains to be considered "converged".

```r
source("../Summary Figures/visualize_sims.R")
visualize_parameter_group(sim_summary = sims_output,
                          pars = "lambda",
                          theta_scenario = 1,
                          scenarios = 1:4,
                          convergence_threshold = 1.1) # for illustration only!
```



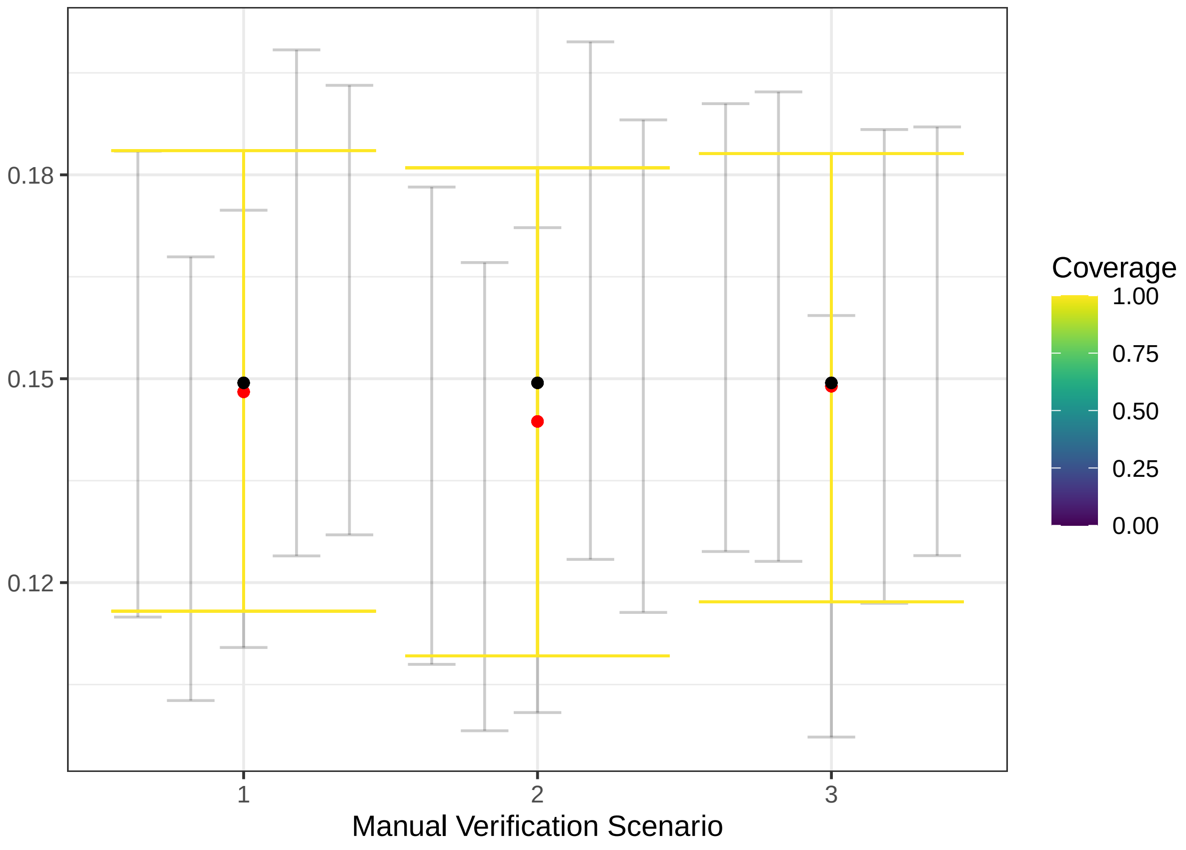
The features of the plot are as follows:

- Facet grids: parameters
- X-axis: Manual verification scenario
- y-axis: parameter values
- Small grey error bars: 95% posterior interval for an individual model fit where all parameters were below `convergence_threshold`.
- Colored error bars: average 95% posterior interval across all converged models under that scenario.
- Color: Coverage, or the rate at which 95% posterior intervals contain the true data-generating parameter value.

- Black dots: the true value of the parameter
- Red dots: average posterior mean

If you would like to visualize a single parameter, use `visualize_single_parameter`, which takes the same arguments as the previous visualization function:

```r
# note the space between the indices for theta[2, 1]
visualize_single_parameter(sims_output, par = "theta[2, 1]",
                           theta_scenario = 1,
                           scenarios = 1:3,
                           convergence_threshold = 1.2)
```



Note that the scale of the y-axis is free to change from one visualization to the next. Additionally, if no datasets show evidence of convergence (i.e., no fitted models have $\hat{R} \leq c$ for all parameters, where c is the specified convergence threshold) under a given scenario, the scenario will not appear on the x-axis.

3 Example: simulations with a fixed-effort design

The previous section outlined the mechanics of using the functions contained in the ValidationExplorer repo. Here, we provide a streamlined example with a fixed-effort validation design, and emphasize how the tool

Species	ψ	λ
EPFU	0.633	5.934
LACI	0.612	4.160
LANO	0.849	14.25
MYLU	0.898	28.25

Table 1: Posterior estimates obtained by Stratton et al., (2022)

may be used to aid the decision process about the appropriate level of effort for reliable inference. Our example uses the second validation design available in the ValidationExplorer: a fixed-effort design. Under this validation design, a random sample of p percent of all recordings is taken from the first detector night at each site, where p is specified by the user.

We begin by selecting the parameter values for each species (i.e., assumed occurrence probability and relative activity rate). If estimates exist for the species of interest, one option could be to adopt those. For instance, if we assumed an assemblage comprised of *Eptesicus fuscus* (EPFU), *Lasiurus cinereus* (LACI), *Lasinoycteris noctivagans* (LANO), and *Myotis lucifugus* (MYLU), we could use posterior estimates obtained by Stratton et al. (2022). These are shown in Table 1.

Note that the `scenarios` argument is now a vector, in contrast with the proceeding example.

```r
FE_data <- simulate_validatedData(
  n_datasets = 2,
  validation_design = "FixedPercent",
  scenarios = c(0.05, .25,  0.5, 0.75), # Note the vector of possible scenarios
  nsites = nsites,
  nvisits = nvisits,
  nspecies = nspecies,
  psi = psi,
  lambda = lambda,
  theta = test_theta2,
  save_datasets = TRUE,
  save_masked_datasets = TRUE,
  directory = paste0(here::here("Testing", "FixedEffortExample"))
)
```

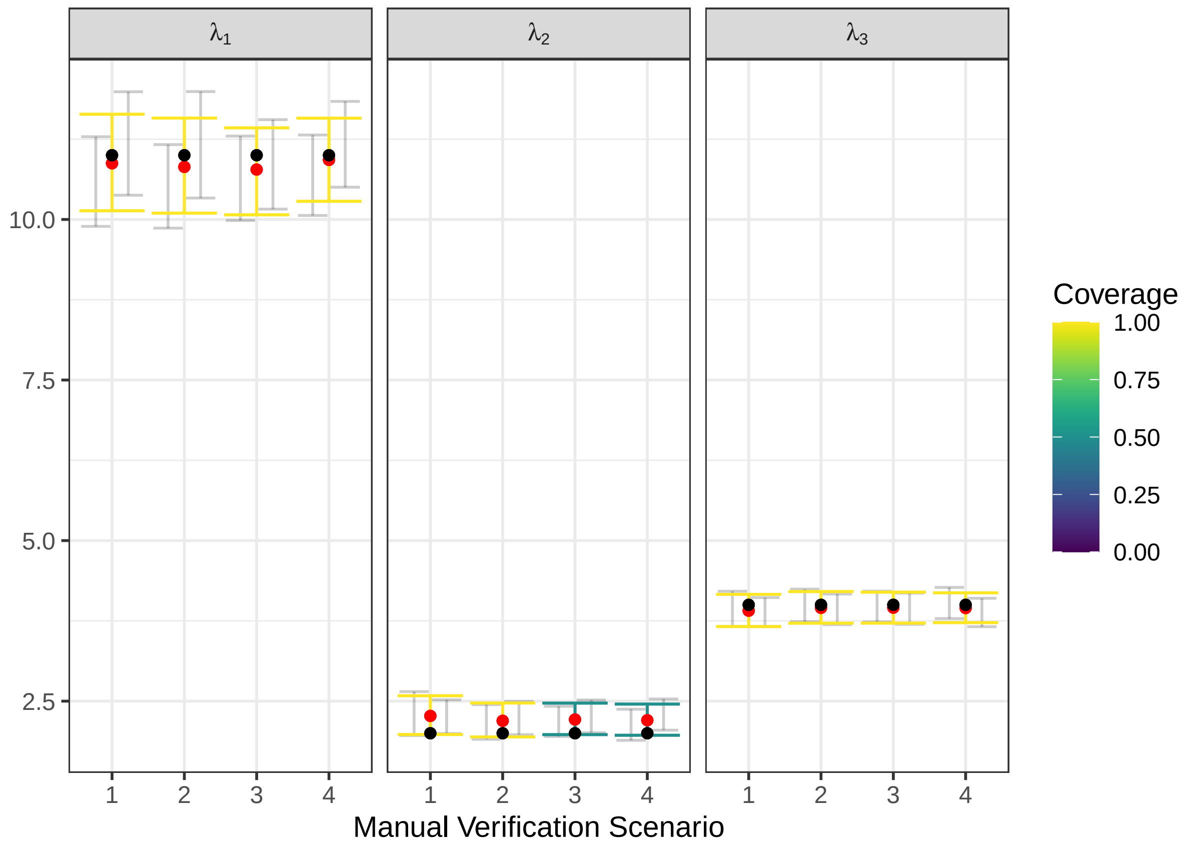
```
## ‘summarise()‘ has grouped output by ’site’, ’visit’, ’true_spp’. You can
## override using the ‘.groups‘ argument.
## ‘summarise()‘ has grouped output by ’site’, ’visit’, ’true_spp’. You can
## override using the ‘.groups‘ argument.
```

Next, we fit the model to the simulated data:

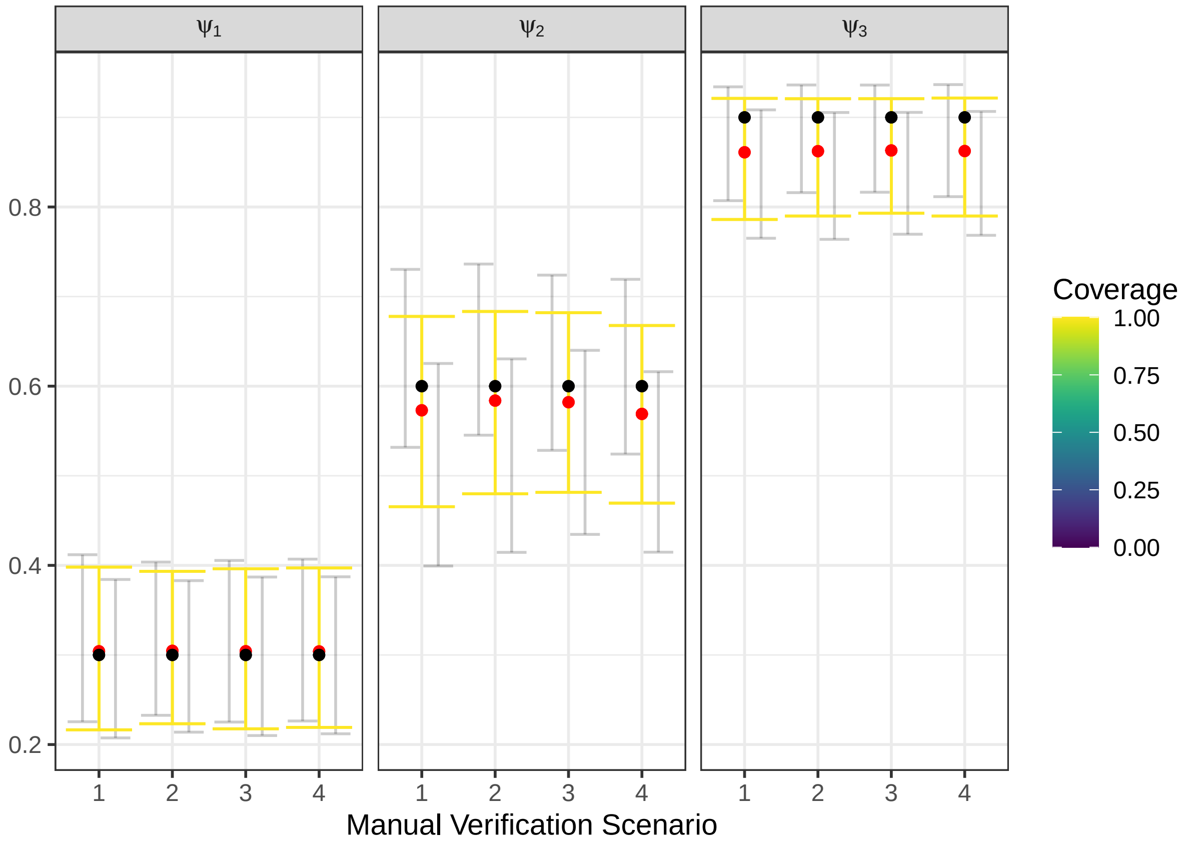
```r
FE_model_fits <- run_sims(
  data_list = FE_data$masked_dfs,
  zeros_list = FE_data$zeros,
  theta_scenario_id = 1,
  save_fits = FALSE,
  DGVs = list(lambda = lambda, psi = psi, theta = test_theta2),
  save_individual_summaries_list = TRUE,
  directory = here("Testing", "FixedEffortExample")
)
```

We can visualize the results from the model for each parameter group:

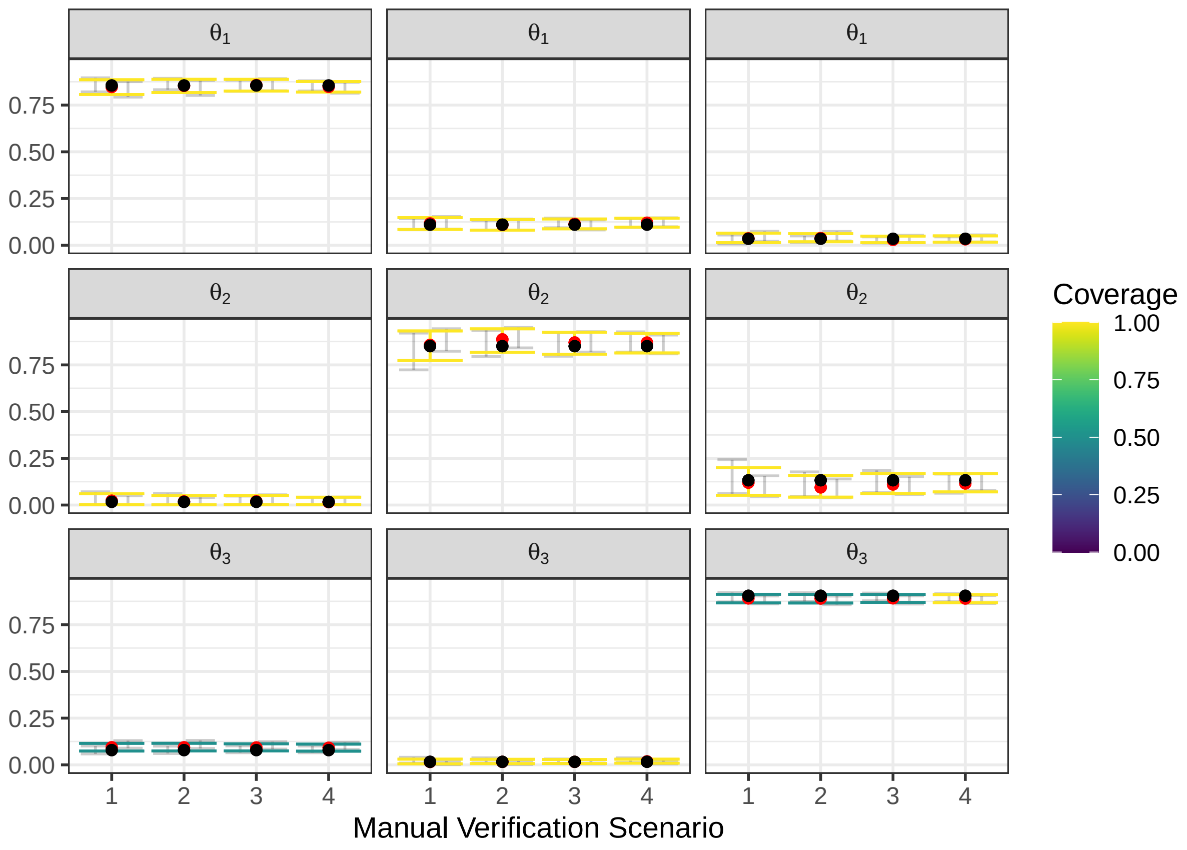
```r
visualize_parameter_group(FE_model_fits, pars = "lambda", theta_scenario = 1, scenarios = 1:4)
```



```r
visualize_parameter_group(FE_model_fits, pars = "psi", theta_scenario = 1, scenarios = 1:4)
```



```r
visualize_parameter_group(FE_model_fits, pars = "theta", theta_scenario = 1, scenarios = 1:4)
```



Note the number of recordings simulated under each scenario:

```r
summarize_n_validated(FE_data$masked_dfs)
```

```
## [1]  99.0 235.5 424.5 640.5
```

Note that inference is only

Banner, Katharine M., Kathryn M. Irvine, Thomas J. Rodhouse, Wilson J. Wright, Rogelio M. Rodriguez, and Andrea R. Litt. 2018. "Improving Geographically Extensive Acoustic Survey Designs for Modeling Species Occurrence with Imperfect Detection and Misidentification." *Ecology and Evolution* 8 (12): 6144–56. https://doi.org/https://doi.org/10.1002/ece3.4162.

de Valpine, Perry, Christopher Paciorek, Daniel Turek, Nick Michaud, Cliff Anderson-Bergman, Fritz Obermeyer, Claudia Wehrhahn Cortes, Abel Rodrìguez, Duncan Temple Lang, and Sally Paganin. 2022. *NIMBLE User Manual* (version 0.12.2). https://doi.org/10.5281/zenodo.1211190.

Irvine, Kathryn M., Katharine M. Banner, Christian Stratton, William M. Ford, and Brian E. Reichert. 2022. "Statistical Assessment on Determining Local Presence of Rare Bat Species." *Ecology*.

Loeb, Susan C., Thomas J. Rodhouse, Laura E. Ellison, Cori L. Lausen, Jonathan D. Reichard, Kathryn

M. Irvine, Thomas E. Ingersoll, et al. 2015. “A Plan for the North American Bat Monitoring Program (NABat).” SRS-208. USDA Forest Service.

Stratton, Christian, Kathryn M. Irvine, Katharine M. Banner, Wilson J. Wright, Cori Lausen, and Jason Rae. 2022. “Coupling Validation Effort with in Situ Bioacoustic Data Improves Estimating Relative Activity and Occupancy for Multiple Species with Cross-Species Misclassifications.” *Methods in Ecology and Evolution* 13 (6): 1288–1303. https://doi.org/https://doi.org/10.1111/2041-210X.13831.

Valpine, Perry de, Daniel Turek, Christopher J. Paciorek, Clifford Anderson-Bergman, Duncan Temple Lang, and Rastislav Bodik. 2017. “Programming With Models: Writing Statistical Algorithms for General Model Structures With NIMBLE.” *Journal of Computational and Graphical Statistics* 26 (2): 403–13. https://doi.org/10.1080/10618600.2016.1172487.

Wright, Wilson J., Kathryn M. Irvine, Emily S. Almberg, Andrea R. Litt, and Nigel Yoccoz. 2020. “Modelling Misclassification in Multi-species Acoustic Data When Estimating Occupancy and Relative Activity.” *Methods in Ecology and Evolution* 11 (1): 71–81.