Vignette: Simulation studies with `ValidationExplorer`

2025-03-05

Abstract

Our vignette demonstrates the use of the `ValidationExplorer` package to conduct statistical simulation studies that can aid in the design phase of bioacoustic studies. In particular, our package facilitates explorion of the costs and inferential properties (e.g., coverage and interval widths) of alternative validation designs in the context of the count detection model framework. Our functions allow the user to specify a suite of candidate validation designs using either a stratified sampling procedure or a fixed-effort design type. An example of the former is provided in the manuscript entitled '`ValidationExplorer`: Streamlined simulations to provide bioacoustic study design guidance in the presence of misclassification', which was submitted to the Applications series of *Methods in Ecology and Evolution*. In this vignette, we provide further details not covered in the manuscript and an additional example of data simulation, model fitting, and visualization of simulation results when using a fixed-effort design type. 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.

Contents

1 Introduction

Automated recording units (ARUs) provide one of the main data sources for many contemporary ecological studies that aim to provide inference about status and trends for assemblages of species (Loeb et al. 2015). As described in the main text, entitled, "`ValidationExplorer`: Streamlined simulations to provide bioacoustic study design guidance in the presence of misclassification," substantial practical interest lies in identifying cost-effective validation designs that will allow a study to obtain its measurable objectives. We believe that statistical simulation studies are a valuable tool for informing study design – including the validation step – prior to gathering ARU data, and it is our goal for `ValidationExplorer` to provide those tools in an approachable package.

This vignette aids exploration of possible validation designs within a count-detection framework using the `ValidationExplorer` package. As described in the main text, a validation design is composed of two parts: a random mechanism for selecting observations to be manually reviewed by experts ("validated"), and a percentage or proportion that controls the number of validated recordings. We refer to the random mechanism as the *design type* and the proportion as the level of validation effort (LOVE). In the following section we consider five possible LOVEs and show an example of a fixed effort design.

We emphasize that results from any simulation study – including those produced using the `ValidationExplorer` package – are conditional on the settings and assumptions of the study. In the case of the count-detection model framework that is implemented in `ValidationExplorer`, the assumptions are

1. The occurrence of species within a site are independent; the presence of one species carries no information about the presence or absence of another.
2. For any one species, its occurrence at one location is independent of its occurrence at any other location (independence across sites).
3. Visits to a site (i.e., detector nights) are independent.
4. Recordings within the same site-visit are independent.
5. Each recording is assigned a single species autoID.
6. Validation of one recording does not influence the probability that another will be validated.
7. At sites where at least one species is present, the count of recordings generated by each species per night is a Poisson random variable.

8. All species in the assemblage can co-occur and are capable of being confused (no structural zeros in the classification matrix).

9. The configuration of visits to sites is balanced. Note that this is not required to use our method but it is assumed in the simulation studies. For example, the data used by Oram et al. (in submission) had a varying number of detector nights at each site. To model these data, Oram et al. (in submission) made the additional assumption that the number of visits to a site is independent of the presence/absence and expected relative activity of species there. That is, locations where detectors were deployed for more nights were not more or less likely than other sites to be occupied by species of interest and were not more or less likely than other sites to have high levels of activity.

10. The fitted model is the same as the model that generated the data.

Under these assumptions, we outline the step-by-step use of `ValidationExplorer` in the following section.

2 Conducting a simulation study with a fixed effort design type

2.1 What is a fixed effort design?

In this section, we assume a fixed-effort design type, under which $x\%$ of recordings obtained from each visit to a site are validated by experts. The level of validation effort is controlled by the value of x. We begin the process of simulating under this design by defining the real-world objectives and constraints we anticipate.

2.2 Step 0: Define measurable objectives and constraints

Recall from the main text that the first step – before opening R and loading `ValidationExplorer` – is to identify and write down the set of measurable objectives that the data will be used for. Suppose that, for this example, the measurable objectives and cost constraints are the same as those in Section 3 of the main text. That is,

- The measurable objective is to estimate relative activity parameters (denoted as λ_k) for each species with 95

- The monitoring program can pay their expert bat biologists to validate at most 4000 recordings. We make the assumption that all recordings are approximately the same cost to validate (i.e., rare species autoIDs are not necessarily more expensive or time consuming than extremely common ones).

Table 1: Prior knowledge of relative activity rates and occurrence probabilities for the six bat species of interest. These values will be used to simulate data.

Species	ψ	λ
Eptesicus fuscus (EPFU)	0.63	5.9
Lasiurus cinereus (LACI)	0.61	4.2
Lasionycteris noctivagans (LANO)	0.85	14.3
Myotis californicus (MYCA)	0.70	6.2
Myotis ciliolabrum (MYCI)	0.24	11.9
Myotis evotis (MYEV)	0.70	2.4

Suppose further that the species assemblage is the same as in the main text. That is, we have six species of interest that co-occur. The existing prior knowledge (perhaps from another study) about the relative activity rates and occurrence probabilities for each species are summarized in Table 1.

2.3 Step 1: Installing and loading required packages

Once measurable objectives and constraints are clearly defined, the next step is to load the required packages. For first time users, it may be necessary to install a number of dependencies, as shown by Table 1 in the main text. If you need to install a dependency or update the version, run the following, with `your_package_name_here` replaced by the name of the package:

```
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)
```

Finally, install and load `ValidationExplorer` by running

```
devtools::install_github(repo = "j-oram/ValidationExplorer")
library(ValidationExplorer)
```

With `ValidationExplorer` installed, users have access to all the functions outlined in the table of functions found in Section 5.

2.4 Step 2: Simulate data

The first step in a simulation study is to simulate data under each of the candidate validation designs, which is accomplished with the `simulate_validatedData` function in `ValidationExplorer`. We begin by assigning values for the number of sites, visits, and species, as well as the parameter values in Table 1, which are existing estimates obtained by Stratton et al. (2022):

```
# Set the number of sites, species and visits
nsites <- 30
nspecies <- 6
nvisits <- 4

psi <- c(0.6331, 0.6122, 0.8490, 0.6972, 0.2365, 0.7036)
lambda <- c(5.9347, 4.1603, 14.2532, 6.1985, 11.8649, 2.4050)
```

Note that by running multiple simulation studies with varying numbers of sites and balanced visits, `ValidationExplorer` allows users to investigate all elements of the study design prior to data collection. In addition to specifying the numnber of sites and visits and parameter values, `simulate_validatedData` requires that the user supply misclassification probabilities in the form of a matrix, subject to the constraint that rows in the matrix sum to one. An easy way to simulate a matrix of probabilities that meet these criteria is to leverage the `rdirch` function from the `nimble` package:

```
# Simulate a hypothetical confusion matrix
set.seed(10092024)
Theta <- t(apply(diag(29, nspecies) + 1, 1, function(x) {nimble::rdirch(alpha = x)}))
```

Note that the above definition of `Theta` places high values on the diagonal of the matrix, corresponding to a high probability of correct classification. To lower the diagonal values, change the specification of `diag(29, nspecies)` to a smaller value. For example:

```
another_Theta <- t(apply(diag(5, nspecies) + 1, 1, function(x) {
    nimble::rdirch(alpha = x)
  }))
```

`another_Theta` has lower values on the diagonal, and greater off-diagonal values (i.e., higher probability of misclassification). If you have specific values you would like to use for the assumed classification probabilities (e.g., from an experiment), these can be supplied manually:

```r
manual_Theta <- matrix(c(0.9, 0.05, 0.01, 0.01, 0.02, 0.01,
                         0.01, 0.7, 0.21, 0.05, 0.02, 0.01,
                         0.01, 0.01, 0.95, 0.01, 0.01, 0.01,
                         0.05, 0.05, 0.03, 0.82, 0.04, 0.01,
                         0.01, 0.015,  0.005,  0.005, 0.95, 0.015,
                         0.003, 0.007, 0.1, 0.04, 0.06, 0.79),
                         byrow = TRUE, nrow = 6)

print(manual_Theta)
```

```
##       [,1]  [,2]  [,3]  [,4] [,5]  [,6]
## [1,] 0.900 0.050 0.010 0.010 0.02 0.010
## [2,] 0.010 0.700 0.210 0.050 0.02 0.010
## [3,] 0.010 0.010 0.950 0.010 0.01 0.010
## [4,] 0.050 0.050 0.030 0.820 0.04 0.010
## [5,] 0.010 0.015 0.005 0.005 0.95 0.015
## [6,] 0.003 0.007 0.100 0.040 0.06 0.790
```

If you define the classifier manually, make sure the rows sum to 1 by running

```r
all(rowSums(manual_Theta) == 1) # want this to return TRUE
```

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

```r
# If the above returns FALSE, see which one is not 1:
rowSums(manual_Theta)
```

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

With the required inputs defined, we can simulate data:

```r
sim_data <- simulate_validatedData(
    n_datasets = 10, # For demonstration -- use 50+ for real simulation studies
    nsites = nsites,
    nvisits = nvisits,
    nspecies = nspecies,
    design_type = "FixedPercent",
    scenarios = c(0.05, 0.1, 0.15, 0.3),
    psi = psi,
    lambda = lambda,
    theta = Theta,
    save_datasets = FALSE, # default value is FALSE
    save_masked_datasets = FALSE, # default value is FALSE
    directory = here::here("Vignette", "Fixed_Effort")
)
```

Note that we specified the design type through the argument `design_type = "FixedPercent"`, with the possible scenarios defined as the vector `scenarios = c(0.05, 0.1, 0.15, 0.3)`. These two arguments

specify the set of alternative validation designs we will compare in our simulation study. Under the first validation design, 5% of recordings from each visit to a site are validated, while in the second validation design 10% of recordings from each visit to a site are validated, and so on.

To understand the output from `simulate_validatedData`, we can investigate `sim_data`. The output is a list, containing three objects:

```
names(sim_data)
```

```
## [1] "full_datasets" "zeros"         "masked_dfs"
```

We examine each of these objects below:

- `full_datasets`: A list of length `n_datasets` with unmasked datasets. These are the datasets under complete validation so that every recording has an autoID and a true species label. We opted to not save these datasets to the working directory by setting `save_datasets = FALSE`. If we had specified `save_datasets = TRUE`, then these will be saved individually in `directory` as `dataset_n.rds`, where `n` is the dataset number. As an example of one element in `sim_data$full_datasets`, we examine the third simulated full dataset:

```
full_dfs <- sim_data$full_datasets
head(full_dfs[[3]]) # Dataset number 3 if all recordings were validated
```

```
## # 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   5.93 0.633 0.954      1     4    24
## 2     1     1        1      1   5.93 0.633 0.954      1     4    24
## 3     1     1        1      1   5.93 0.633 0.954      1     4    24
## 4     1     1        1      1   5.93 0.633 0.954      1     4    24
## 5     1     1        4      2   6.20 0.697 0.0289     1     1    24
## 6     1     1        3      3  14.3  0.849 0.826      1    10    24
```

Notice that in addition to the site, visit, true species and autoID (`id_spp`) columns, the parameter values (`lambda`, `psi`, and `theta`) are given for each true species-autoID combination. In addition, the occupancy state `z` for the true species is given and the `count` of calls at that site visit with a specific true species-autoID label combination. For example, at site 1, visit 1, there are four calls from species 1 that are assigned autoID 1, yielding 4 rows with `count = 4`. There is also one call from species 4 that was assigned a species 2 label with probability 0.02893559. Because this happened once, it is documented with `count = 1` and only

occupies a single row. Next, we can see that there were 10 calls that were correctly identified as species 3; ten rows will have `true_spp = 3` and `autoID = 3`. Finally, the `Y.` column tells us how many observations were made from all species at that site visit; for visit 1 to site 1, the unique value is 24. That is, the 25th row of this dataset will contain the first observation from visit 2 to site 1.

- `zeros`: A list of length `n_datasets` containing the true species-autoID combinations that were never observed at each site visit. These zero counts are necessary for the model to identify occurrence probabilities and relative activity rates. The `count` column, which, again, contains the count of each site-visit-true_spp-id_spp combination, is zero for all entries. For example, in dataset 3 (below), species 2 was not present at site 1, so it could not be classified as species 1 (first row). Additionally, species 3 was present at site 1, but it was never classified as species 1 on visit 1 (second row). If `save_datasets = TRUE`, the zeros for each dataset will also be saved in `directory` individually as `zeros_in_dataset_n.rds`, where `n` is the dataset number.

```
zeros <- sim_data$zeros

# The site-visit-true_spp-autoID combinations that were never observed in
# dataset 3. Notice that count = 0 for all rows!
head(zeros[[3]])
```

```
## # 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      1   4.16 0.612 0.0286      0     0    24
## 2     1     1        3      1  14.3  0.849 0.0137      1     0    24
## 3     1     1        4      1   6.20 0.697 0.0426      1     0    24
## 4     1     1        5      1  11.9  0.236 0.00559     0     0    24
## 5     1     1        6      1   2.40 0.704 0.00312     1     0    24
## 6     1     1        1      2   5.93 0.633 0.0159      1     0    24
```

- `masked_dfs`: A nested list containing each dataset masked under each scenario. For example, `masked_dfs[[4]][[3]]` contains dataset 3, assuming that it was validated according to scenario 4 (30% of recordings randomly sampled from each site-visit for validation). 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.

```
masked_dfs <- sim_data$masked_dfs

# View dataset 3 subjected to the validation design in scenario 4:
# randomly select and validate 30% of recordings from the first visit
```

```
# to each site
head(masked_dfs[[4]][[3]], 10)
```

```
## # A tibble: 10 x 12
##     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       NA      1   5.93 0.633 0.954      1     4    24
##  2     1     1       NA      1   5.93 0.633 0.954      1     4    24
##  3     1     1       NA      1   5.93 0.633 0.954      1     4    24
##  4     1     1       NA      1   5.93 0.633 0.954      1     4    24
##  5     1     1       NA      2   6.20 0.697 0.0289     1     1    24
##  6     1     1        3      3  14.3  0.849 0.826      1    10    24
##  7     1     1        3      3  14.3  0.849 0.826      1    10    24
##  8     1     1        3      3  14.3  0.849 0.826      1    10    24
##  9     1     1       NA      3  14.3  0.849 0.826      1    10    24
## 10     1     1        3      3  14.3  0.849 0.826      1    10    24
## # i 2 more variables: unique_call_id <chr>, scenario <int>
```

The name of this nested list comes from the way that validation effort is simulated: recordings that are not selected for validation have their `true_spp` label masked with an `NA`. Notice that in the example output, all entries in the dataset are identical to the unmasked version output in `full_dfs` above, with the exception of the `true_spp` column. From this column we can see that calls 1-5 and 9 were not selected for validation (because `true_spp = NA`), while recordings 6-8 and 10 were (because the true species label is not marked as `NA`).

2.4.1 Summarize validation effort

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:

```
summarize_n_validated(
  data_list = sim_data$masked_dfs,
  theta_scenario = "1",
  scenario_numbers = 1:4
)
```

```
## # A tibble: 4 x 3
##   theta_scenario scenario n_validated
##   <chr>          <chr>          <dbl>
## 1 1              1               218.
## 2 1              2               375.
## 3 1              3               540.
## 4 1              4              1021.
```

We can see here that any of the validation designs considered in our simulations will remain well within budget.

2.5 Step 3: MCMC tuning

Running a complete simulation study can be time consuming. In an effort to help users improve the efficiency of their simulations, we provide the `tune_mcmc` function, which outputs information about possible values for the warmup and total number of iterations required for the MCMC to reach approximate convergence. This function takes in a masked dataset and the corresponding zeros, fits a model to these data, and outputs an estimated run time for 10,000 iterations, as well as the estimated number of required warmup and total iterations. These are intended to assist tuning of the MCMC algorithm, which is done by the user in the following steps, which we walk through in greater detail below:

1. Use `tune_mcmc` to fit a model with multiple long chains.
2. Create trace plots for all model parameters.
3. Examine effective sample sizes n_{eff} and Gelman-Rubin statistics $\hat{R}$ for all parameters.
4. Choose values for the number of iterations and warmup that are slightly larger than what is needed based on steps 1-3. This may help ensure that a greater number of model fits are available to inform simulation study results.

2.5.1 Fit a model

As in the main text, we use a dataset from the scenario with the lowest number of validated recordings, as we expect the greatest number of iterations for this scenario. In our example, this is scenario 1 , in which an average of ≈ 218 recordings are validated per dataset (Section 2.4.1).

```
scenario_number <- 1
dataset_number <- sample(1:length(masked_dfs[[scenario_number]]), 1)

tune_list <- tune_mcmc(
  dataset = sim_data$masked_dfs[[scenario_number]][[dataset_number]],
  zeros = sim_data$zeros[[dataset_number]]
)
```

```
## [1] "Fitting MCMC in parallel ... this may take a few minutes"
```

The output from `tune_mcmc` is a list containing draws from the fitted model, the time required to fit the model with 10,000 draws, MCMC diagnostics and guesses for the number of iterations and warmup required to reliably fit a model. If the guessed values for total iterations and/or warmup are greater than 10,000 draws, an error is issued. We can see the names for each object by running the following block:

```r
names(tune_list)
```

```
## [1] "max_iter_time"    "min_warmup"       "min_iter"         "fit"
## [5] "MCMC_diagnostics"
```

The first element is the time required to fit a model with three chains of 10,000 iterations each:

```r
tune_list$max_iter_time
```

```
## Time difference of 3.173303 mins
```

This may seem insignificant, but over the course of an entire simulations study with 5 scenarios $\times$ 50 datasets, that corresponds to around 8 hours of run time. Using fewer than 10,000 iterations will substantially reduce the time to run a simulation study.

2.5.2 Create trace plots

To decide the number of iterations and warmup for use in a simulation study, we recommend beginning by creating trace plots, which show how the sampled values of a parameter evolve over the course of each Markov chain. To ensure the MCMC algorithm will characterize the posterior distribution well, we need to check that chains are stationary and mixing well, and that effective sample sizes are sufficiently large to characterize the posterior. Trace plots are especially useful for assessing the first of these. To create a trace plot using the bayesplot package (Gabry et al. 2019) for a single parameter, run the following:

```r
# Load bayesplot package specifically designed for visualizing
library(bayesplot)
```

```
## This is bayesplot version 1.11.1
```

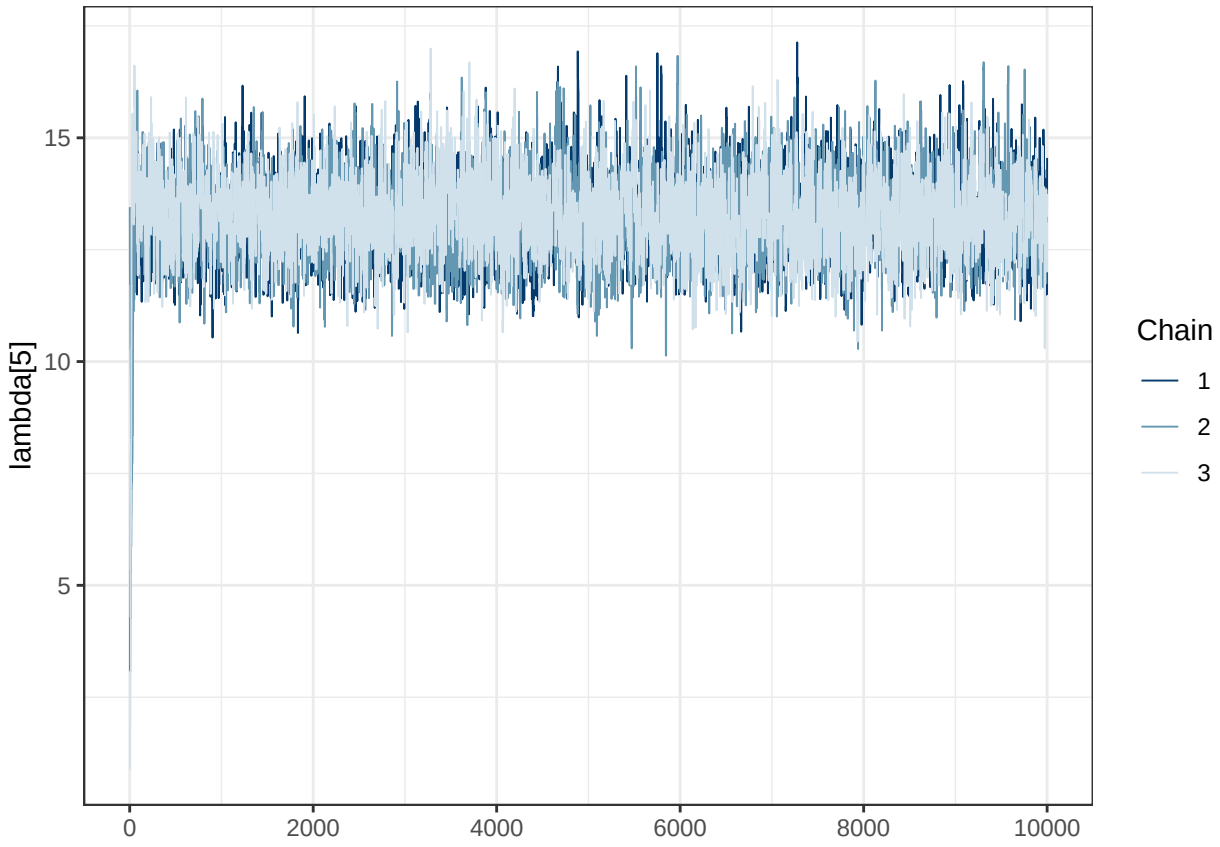
```
## - Online documentation and vignettes at mc-stan.org/bayesplot
```

```
## - bayesplot theme set to bayesplot::theme_default()
```

```
##    * Does _not_ affect other ggplot2 plots
```

```
##    * See ?bayesplot_theme_set for details on theme setting
```

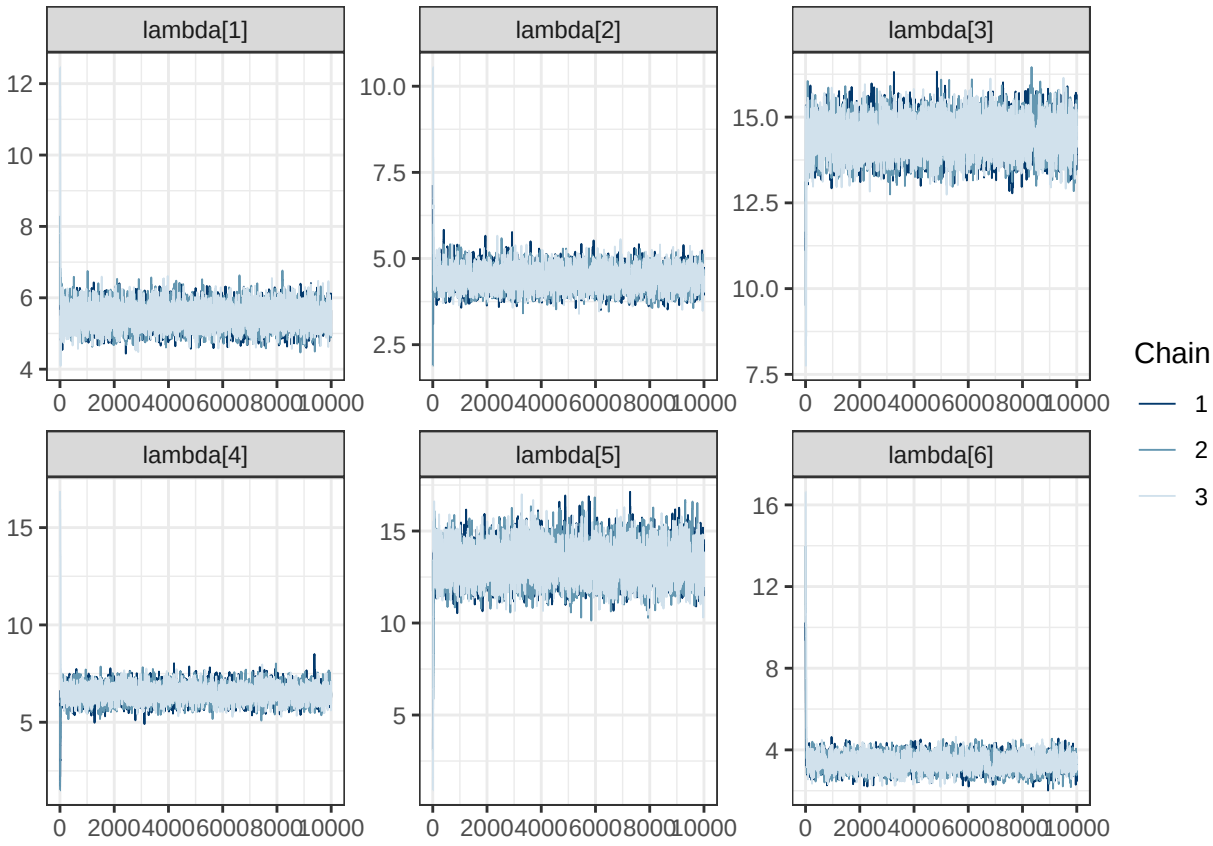
```
# extract the fitted model from tune_mcmc output
fit <- tune_list$fit

# create traceplot
mcmc_trace(fit, pars = "lambda[5]")
```



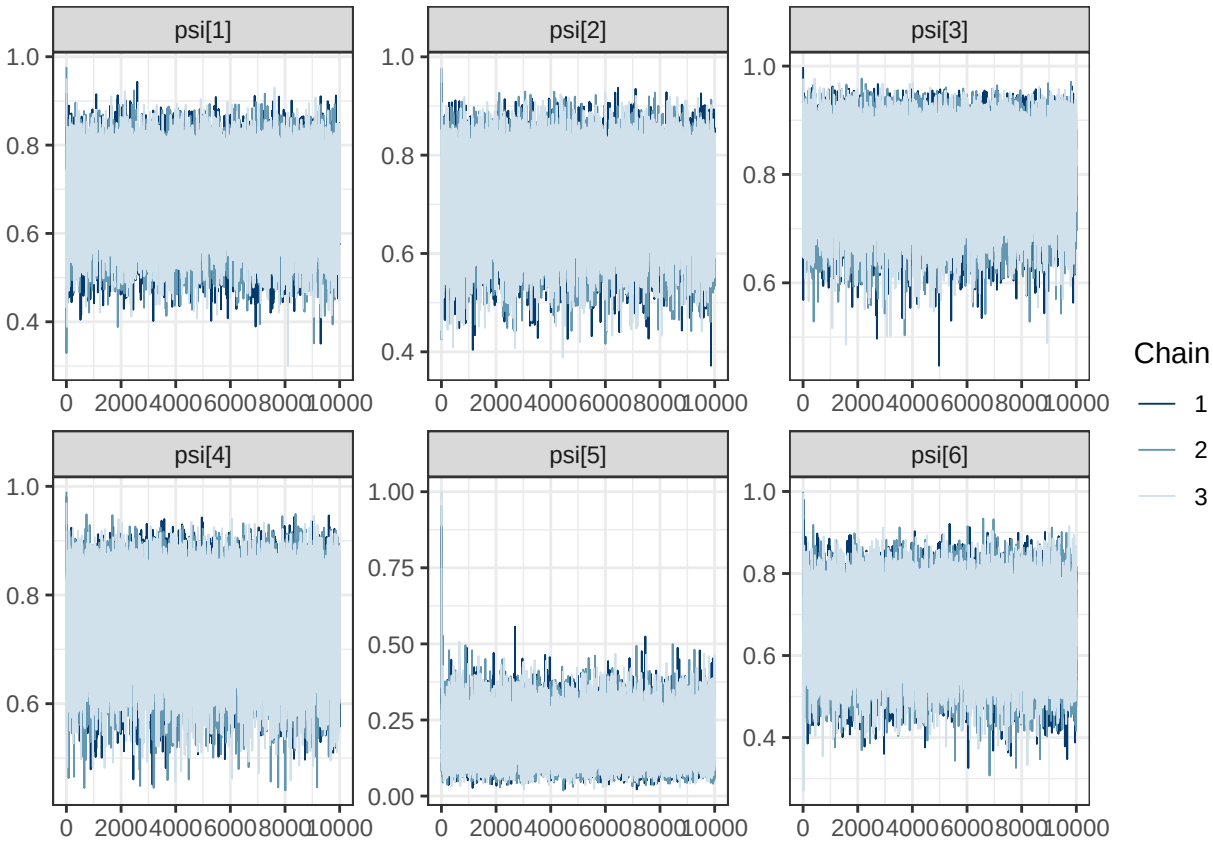
We can see that far fewer than 10,000 iterations are likely required because the chains are mixing well after a few hundred iterations. This is shown by strongly overlapping chains, with no one chain appearing by itself in a region of the parameter space. Chains also appear to be stationary because they do not wander vertically substantially after the first few hundred iterations.

We need to check that chains are stationary and mixing for all parameters. One way to accelerate visual inspection this is through the `regex_pars` argument, which shows relative activity parameters for all species as in the following three code blocks.

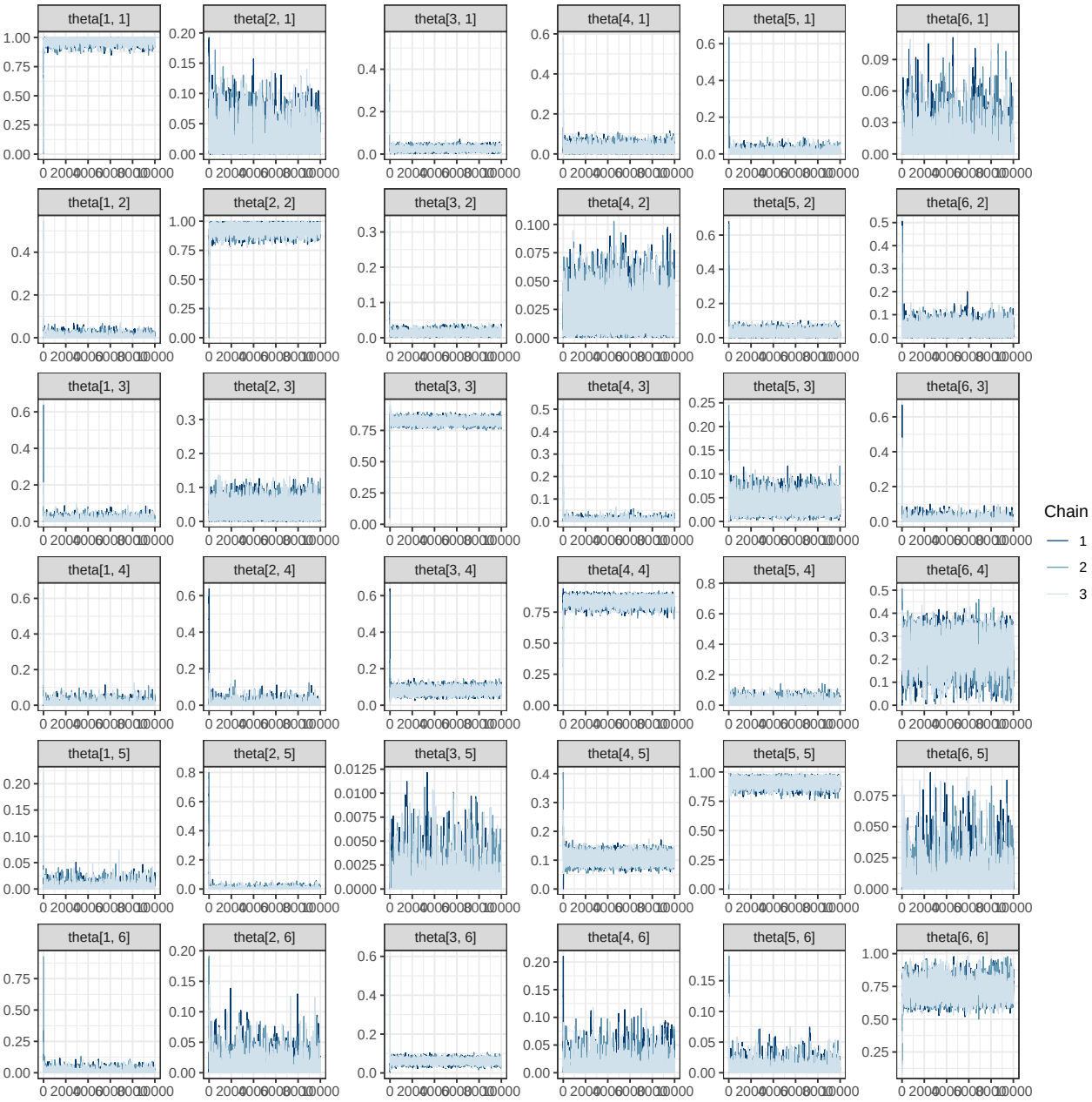
```
mcmc_trace(fit, regex_pars = "lambda")
```




```
mcmc_trace(fit, regex_pars = "psi")
```




```
# create a traceplot for all elements of the confusion matrix
mcmc_trace(fit, regex_pars = "theta")
```



In all of the trace plots for all of the parameters, chains appear to mix quickly, meaning that it is possible a far smaller number of iterations may be suitable for a simulation study. A good place to start for reducing the number of iterations is from the output given by `tune_mcmc`. In our case, we can see that a guess for the minimum number of iterations is 1500 with a warmup of 500:

```
tune_list$min_iter
```

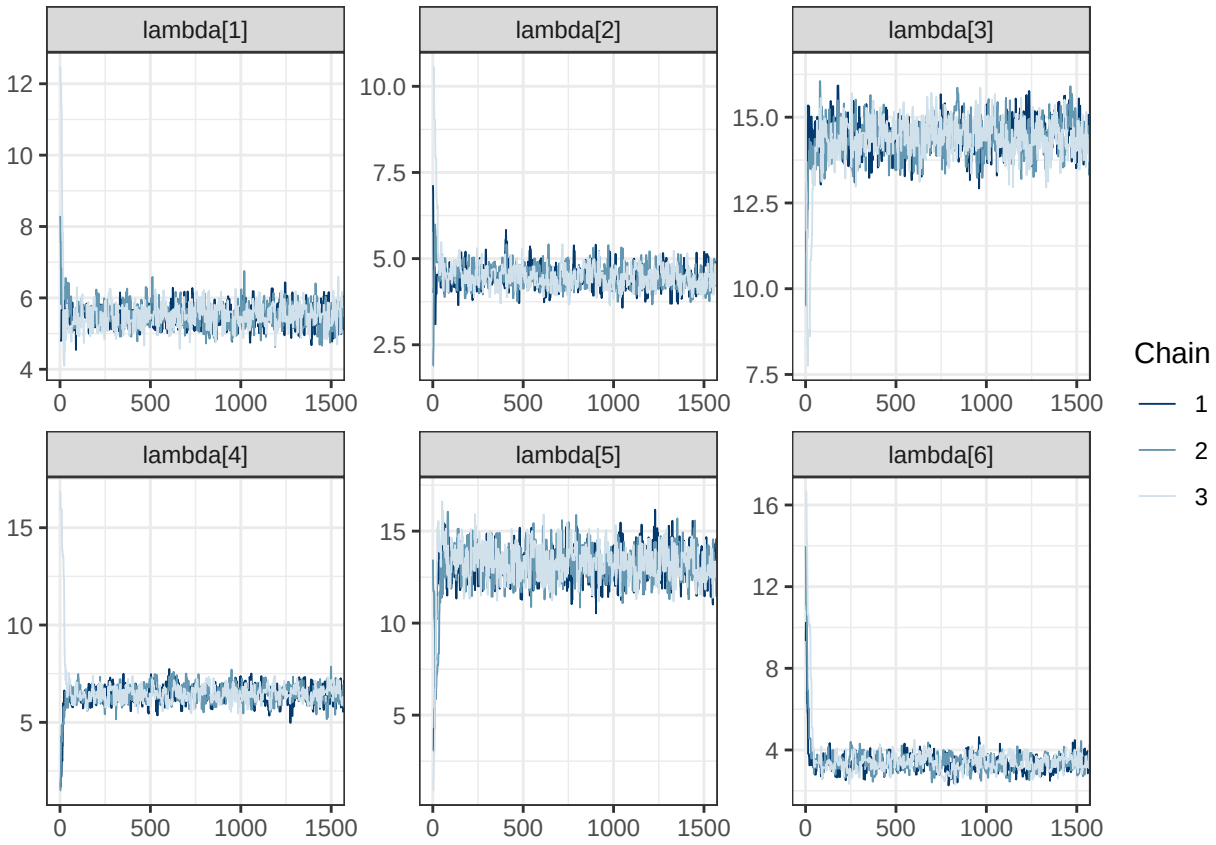
```
## [1] 1500
```

```
tune_list$min_warmup
```

```
## [1] 500
```

It is possible to use these values to zoom in on the trace plots by using the `window` argument:

```
mcmc_trace(fit, regex_pars = "lambda", window = c(0, tune_list$min_iter))
```



Once again, we see the three chains overlap strongly after around 500 iterations and sample around a horizontal line.

After making traceplots and zooming in for all parameters (not all plots are shown here), it appears that the guessed warmup value of 500output from `tune_mcmc` is a reasonable choice. Even so, we increase our iterations and warmup beyond these minimum values in Section 2.6 to avoid simulations failing due to convergence.

2.5.3 Examine effective sample size and Gelman-Rubin statistics

As a final step, we examine the effective sample sizes (out of 10,000 draws) and $\hat{R}$ values. The effective sample size statistics `ess_bulk` and `ess_tail` are MCMC diagnostic statistics that summarizes the number

of effectively independent draws from a parameter's posterior distribution the Markov chain contains. If a parameter has a large value in the `ess_bulk` column, then it is likely that inference based on sampled draws will characterize the center of the posterior distribution well. The `ess_tail` column, on the other hand, describes how much information is available about posterior tail probabilities. Once again, we want a large value for `ess_tail`, preferably `ess_tail`≥ 250.

```
tune_list$MCMC_diagnostics
```

```
##      parameter   ess_bulk  ess_tail      Rhat
## 1    lambda[1]  4256.3250  5127.852 1.0006816
## 2    lambda[2]  2800.4694  4473.479 1.0018897
## 3    lambda[3]  2575.2417  4274.216 1.0015129
## 4    lambda[4]  2362.8355  4074.193 1.0024884
## 5    lambda[5]  3782.8025  5499.534 1.0014149
## 6    lambda[6]  1886.3829  3212.188 1.0011712
## 7       psi[1] 30147.6994 28340.663 0.9999976
## 8       psi[2] 30415.0568 29971.363 0.9999655
## 9       psi[3] 28457.2516 27971.720 1.0001195
## 10      psi[4] 29685.7536 29533.867 0.9999960
## 11      psi[5] 28082.2530 28116.007 0.9999820
## 12      psi[6] 20959.3151 23894.310 1.0000279
## 13 theta[1, 1]  3550.0665  4121.081 1.0008268
## 14 theta[2, 1]   932.1523  1135.856 1.0078027
## 15 theta[3, 1]  2203.3407  3127.003 1.0019207
## 16 theta[4, 1]  3478.9348  4942.455 1.0008129
## 17 theta[5, 1]  1912.9919  2324.898 1.0011651
## 18 theta[6, 1]  2211.3528  2398.381 1.0006527
## 19 theta[1, 2]  3132.1191  3475.343 1.0002028
## 20 theta[2, 2]  1830.7290  4009.910 1.0043791
## 21 theta[3, 2]  4021.2501  4926.832 1.0014111
## 22 theta[4, 2]  3752.3278  5143.469 1.0008940
## 23 theta[5, 2]  5002.1262  5568.425 1.0009407
## 24 theta[6, 2]  3769.8656  5028.680 1.0004643
## 25 theta[1, 3]  2820.8618  3711.457 1.0001600
## 26 theta[2, 3]  4224.2290  4671.734 1.0008169
## 27 theta[3, 3]  2584.5399  6189.363 1.0005896
## 28 theta[4, 3]  2748.9275  2973.467 1.0015043
## 29 theta[5, 3]  5197.7539  5895.055 1.0005027
## 30 theta[6, 3]  2774.9185  3622.834 1.0022336
## 31 theta[1, 4]  3310.3171  3409.099 1.0015490
## 32 theta[2, 4]  3169.3394  3614.631 1.0005934
## 33 theta[3, 4]  2253.6705  4834.142 1.0004177
## 34 theta[4, 4]  3192.2743  3727.733 1.0005853
## 35 theta[5, 4]  1099.8717  1764.071 1.0033490
## 36 theta[6, 4]  1689.3348  1789.603 1.0010853
## 37 theta[1, 5]  3480.7314  3853.632 1.0010053
## 38 theta[2, 5]  3308.4148  3925.670 1.0002122
## 39 theta[3, 5]  3704.4867  3952.964 1.0014661
## 40 theta[4, 5]  6116.3486  8366.593 1.0002993
## 41 theta[5, 5]  2165.0654  3856.834 1.0024596
```

```
## 42 theta[6, 5]  3429.7617  3825.051 1.0021946
## 43 theta[1, 6]  2248.5387  2490.716 1.0004318
## 44 theta[2, 6]  2979.7105  3438.988 1.0014827
## 45 theta[3, 6]  2841.2928  3516.816 1.0006448
## 46 theta[4, 6]  1541.6988  1777.138 1.0015657
## 47 theta[5, 6]  2891.1708  3494.868 1.0015273
## 48 theta[6, 6]  1813.7888  2176.988 1.0009921
```

For all parameters, the bulk and tail effective sample sizes are fairly large, meaning that even with fewer than 10,000 draws, we could expect $n_{\text{eff}} \geq 250$, allowing us to characterize both the center and tails of the posteriors for all parameters with these draws. Furthermore, the $\hat{R}$ values for all parameters is near 1. In general, we want values of $\hat{R} \leq 1.1$ for chains to be considered converged. We can double check by recomputing these statistics on shortened chains:

```
# for each chain, extract iterations 1001:2500 for all parameters
shortened <- lapply(fit, function(x) x[1001:2500,])

# summarize the shortened chains and select the effective sample
# size columns. mcmc_sum is an internal function used inside of `run_sims`, but
# we use it here to quickly obtain MCMC diagnostics for each parameter
mcmc_sum(shortened, truth = rep(0, ncol(shortened[[1]]))) %>%
  select(parameter, ess_bulk, ess_tail, Rhat)
```

```
##      parameter  ess_bulk  ess_tail      Rhat
## 1    lambda[1]  584.5122  985.8642 1.0068409
## 2    lambda[2]  495.2714  636.1690 1.0031446
## 3    lambda[3]  503.0785  755.5623 1.0131563
## 4    lambda[4]  426.6898  777.8442 1.0083441
## 5    lambda[5]  703.2883  999.4798 1.0029023
## 6    lambda[6]  318.0268  491.4303 1.0026385
## 7       psi[1] 4525.7828 4328.2766 0.9999024
## 8       psi[2] 4566.3865 4428.4817 1.0004970
## 9       psi[3] 4233.9464 4128.2334 1.0003851
## 10      psi[4] 4536.8279 4364.3883 0.9997074
## 11      psi[5] 4822.3720 4468.2474 1.0012221
## 12      psi[6] 3513.8895 3983.0615 0.9997362
## 13 theta[1, 1]  720.1076  947.1407 1.0034928
## 14 theta[2, 1]  132.3007  234.4093 1.0116404
## 15 theta[3, 1]  224.5149  489.8316 1.0066565
## 16 theta[4, 1]  484.1771  524.8232 1.0087460
## 17 theta[5, 1]  255.8012  391.6426 1.0153679
## 18 theta[6, 1]  554.6647  714.6540 1.0039287
## 19 theta[1, 2]  532.5000  528.1218 1.0076991
## 20 theta[2, 2]  325.3232  868.3360 1.0087195
## 21 theta[3, 2]  592.0362  646.1804 1.0060841
## 22 theta[4, 2]  618.2763  660.9868 1.0094191
## 23 theta[5, 2]  760.9023  874.1548 1.0012293
## 24 theta[6, 2]  585.4676  688.7327 1.0110692
## 25 theta[1, 3]  411.9160  375.0482 1.0050586
```

```
## 26 theta[2, 3]  806.4627 1077.0223 1.0003097
## 27 theta[3, 3]  404.5124  801.3461 1.0116116
## 28 theta[4, 3]  474.9300  486.9276 1.0095173
## 29 theta[5, 3] 1194.9344 1453.5067 1.0019669
## 30 theta[6, 3]  499.1513  617.8979 1.0059634
## 31 theta[1, 4]  508.9897  477.0396 1.0022575
## 32 theta[2, 4]  640.1378  840.1459 1.0025585
## 33 theta[3, 4]  350.7506  955.0499 1.0063965
## 34 theta[4, 4]  418.9380  474.6982 1.0080628
## 35 theta[5, 4]  229.8807  463.9088 1.0078349
## 36 theta[6, 4]  282.9228  357.5697 1.0082364
## 37 theta[1, 5]  485.8739  549.1513 1.0130723
## 38 theta[2, 5]  557.6870  552.6841 1.0017394
## 39 theta[3, 5]  569.4490  743.1043 1.0067633
## 40 theta[4, 5]  982.8985 1541.2174 1.0077476
## 41 theta[5, 5]  409.2128  900.6401 1.0043852
## 42 theta[6, 5]  509.3406  491.1547 1.0024756
## 43 theta[1, 6]  447.2283  495.8114 1.0024081
## 44 theta[2, 6]  485.0169  335.5598 1.0035492
## 45 theta[3, 6]  403.9130  499.7779 1.0115280
## 46 theta[4, 6]  224.5449  283.7931 1.0113306
## 47 theta[5, 6]  451.5926  417.5433 1.0055596
## 48 theta[6, 6]  286.3150  370.5709 1.0062926
```

These results appear satisfactory, with effective sample sizes in both the tail and bulk of the posterior distributions of more than 250 and $\hat{R}$ near 1. Based on the results of MCMC tuning, it appears that using an MCMC with at least 1500 iterations with at least 500 discarded as warmup should to produce good results for our simulation study.

2.5.4 Set iterations for simulation

Based on our findings in the MCMC tuning step, we set the number of iterations for simulation to be slightly higher to guard against convergence issues that preclude using a fitted model for inference:

```r
# to be used in the following section
iters_for_sims <- tune_list$min_iter + 1000
warmup_for_sims <- tune_list$min_warmup + 500
```

2.5.5 A note about non-convergence while tuning

In some instances, we have run `tune_mcmc` with a dataset and received a series of error messages that convergence was not reached in under 10,000 iterations. If this persists after trying to fit several other datasets, we have several options:

1. We could increase the number of iterations in the simulation study to be above 10,000 – perhaps to 20,000 and settle for a longer run time of the simulation study.

2. We could take this as a sign that the level of effort is insufficient to identify model parameters. In this case, this scenario should not be considered.

In our experience fitting these models, the second option seems to often be the case, and we encourage users to remove scenarios from consideration if models fit during the tuning step do not reach convergence within 10,000 iterations. It is possible that all validation scenarios – including ones with very large levels of validation effort – will lead to a lack of convergence if the number of sites and visits are very low, in which case the number of sites and visits should be re-evaluated.

We emphasize that the results from `tune_mcmc` are from a single model fit; they are supplied only as guidelines, and we encourage users to increase the number of iterations above the minimum values output from `tune_mcmc`. While each model fit will take slightly longer with an increased number of total iterations, this approach may save time in the long run by avoiding the need to re-run simulations.

2.6 Step 4: Fit models to simulated data

With the simulated dataset and some informed choices about tuning of the MCMC, we use `run_sims` to run the simulations:

```
sims_output <- run_sims(
         data_list = sim_data$masked_dfs,
         zeros_list = sim_data$zeros,
         DGVs = list(lambda = lambda, psi = psi, theta = Theta),
         theta_scenario_id = "FE", # for "fixed effort"
         parallel = TRUE,
         niter = iters_for_sims,
         nburn = warmup_for_sims,
         thin = 1,
         save_fits = TRUE,
         save_individual_summaries_list = FALSE,
         directory = here::here("Vignette", "Fixed_Effort")
)
```

```
## Beginning scenario 1.
```

```
## 2024-12-28 13:34:33.143357
```

```
##   |                                                                      |
```

```
## Beginning scenario 2.
```

```
## 2024-12-28 13:49:48.485313

##   |                                                                      |

## Beginning scenario 3.

## 2024-12-28 13:58:11.456341

##   |                                                                      |

## Beginning scenario 4.

## 2024-12-28 14:07:01.192863

##   |                                                                      |
```

The output object, `sims_output` is a dataframe with summaries of the MCMC draws for each parameter estimated from each dataset under each validation scenario. Summaries include the posterior mean, standard deviation, Naive standard error, Time-series standard error, quantiles for 50% and 95% posterior intervals, median, and MCMC diagnostics (Gelman-Rubin statistic, and effective samples sizes in the tails and bulk of the distribution).

```
str(sims_output)
```

```
## 'data.frame':    1920 obs. of  19 variables:
##  $ parameter     : chr  "lambda[1]" "lambda[2]" "lambda[3]" "lambda[4]" ...
##  $ Mean          : num  6 4.8 14.05 5.99 11.79 ...
##  $ SD            : num  0.348 0.343 0.449 0.492 0.826 ...
##  $ Naive SE      : num  0.00518 0.00512 0.00669 0.00733 0.01232 ...
##  $ Time-series SE: num  0.0129 0.015 0.019 0.0295 0.032 ...
##  $ 2.5%          : num  5.32 4.19 13.18 5.02 10.27 ...
##  $ 25%           : num  5.77 4.56 13.76 5.66 11.25 ...
##  $ 50%           : num  6 4.79 14.04 5.99 11.7 ...
##  $ 75%           : num  6.23 5.02 14.36 6.3 12.29 ...
##  $ 97.5%         : num  6.71 5.52 14.92 7.01 13.57 ...
##  $ Rhat          : num  1.01 1.01 1 1.02 1.01 ...
##  $ ess_bulk      : num  758 379 561 264 589 ...
##  $ ess_tail      : num  1022 642 1042 562 637 ...
##  $ truth         : num  5.93 4.16 14.25 6.2 11.86 ...
##  $ capture       : num  1 0 1 1 1 1 1 1 1 1 ...
##  $ converge      : num  1 1 1 1 1 1 1 1 1 1 ...
##  $ theta_scenario: chr  "FE" "FE" "FE" "FE" ...
##  $ scenario      : int  1 1 1 1 1 1 1 1 1 1 ...
##  $ dataset       : int  1 1 1 1 1 1 1 1 1 1 ...
```

Note that we fit all models in parallel to reduce simulation time; we encourage users to do the same. However, if you fit models with `parallel = FALSE` in `run_sims`, the console will display the messages NIMBLE prints as it compiles code for each model. Internally, we specify a custom distribution to compute the marginal probabilities for ambiguous autoID labels, and you will see a warning about overwriting a custom user-specified distribution if `parallel = FALSE`. These warnings can be safely ignored.

2.7 Step 5: Visualize simulations

Once models have been fit to all simulated datasets, you can visualize results using several functions. We recommend beginning with the most detailed functions, which are `visualize_parameter_group` and `visualize_single_parameter`. The plots output from these functions show many following features of the simulation study. These are

- Facet grids: parameters (only for `visualize_parameter_group`)
- 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 points: the true value of the parameter
- Red points: average posterior mean

The `visualize_parameter_group` function is useful for examining an entire set of parameters, such as all relative activity parameters. For example, 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.

```
visualize_parameter_group(
  sim_summary = sims_output,
  pars = "lambda",
  theta_scenario = 1,
  scenarios = 1:4,
  convergence_threshold = 1.05
)
```

The output shown in Figure 1 indicates that under all validation scenarios we considered, the expected inference for relative activity rates of species 1-4 and 6 meets our measurable objectives: the posterior

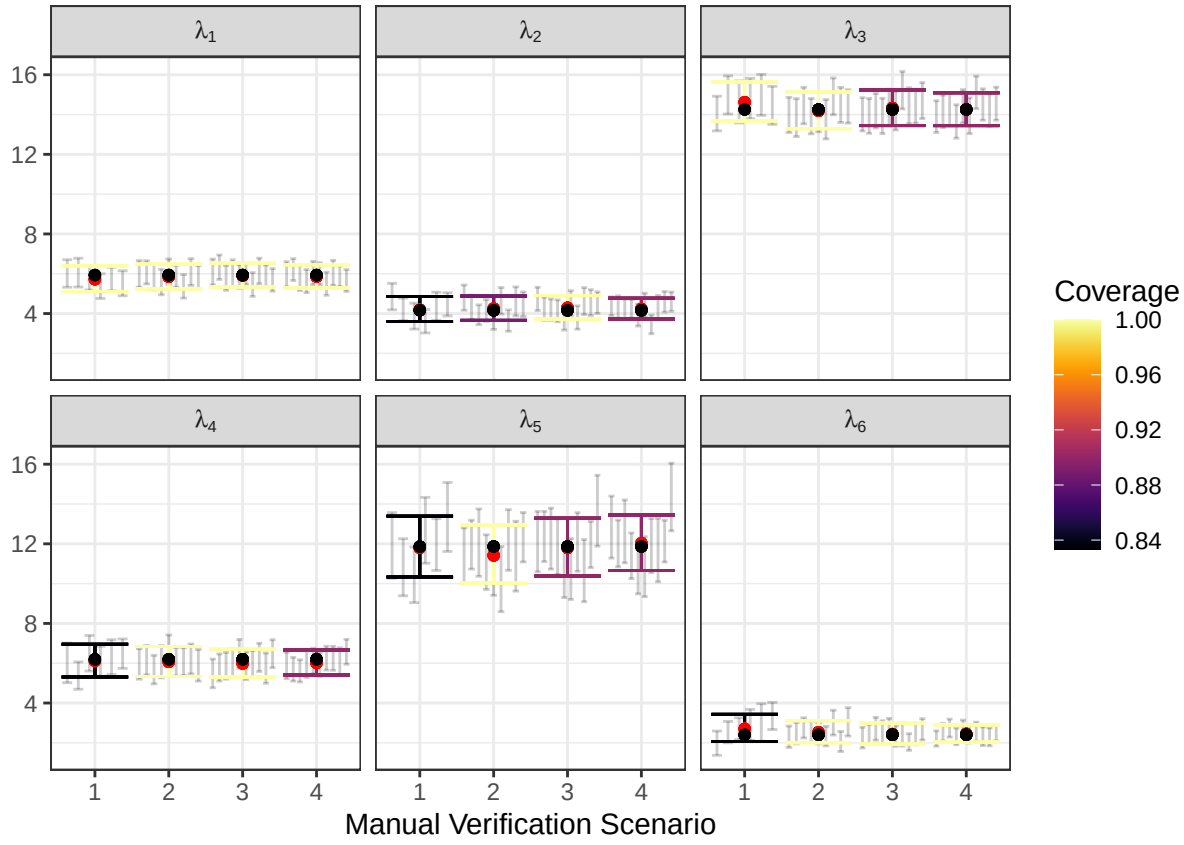


Figure 1: Example output from the `visualize_parameter_group` function. Results are faceted by parameter, with the x-axis indicating the validation scenario number, and the y-axis showing the data generating value (black point). Red points show average posterior means, small grey intervals show 95% posterior intervals from converged model fits and colored intervals are average 95% posterior intervals. Coverage is shown by color, with blue corresponding to low coverage.

interval width is less than three for these species and there is minimal estimation error. However, note that under validation scenario 1 and 2, fewer of the models converged within 2500 iterations of the MCMC, which is visible from smaller number of grey intervals. This suggests that a higher level of validation effort could be beneficial. Additionally, average interval width is never below 3 for species 5; none of the designs considered here meet the measurable objective for this species. We can see this more clearly by examining output through alternative visualization functions.

A first step would be to use `visualize_single_parameter`, which takes the same arguments as the previous visualization function:

```
visualize_single_parameter(
  sims_output, par = "lambda[5]",
  theta_scenario = 1,
  scenarios = 1:4,
  convergence_threshold = 1.05
)
```

The greater detail of `visualize_single_parameter` in Figure 2 underscores the difference in the number of converged models, shown by the number grey 95% posterior interval for each converged model fit. The visualization also helps clarify that while average interval width is near 3 in scenario 2, it is still acceptable for our measurable objectives. Coverage for most relative activity parameters is also near-nominal under this validation scenario, which requires that around 375 recordings be validated in an average dataset. This can also be seen using the `plot_width_vs_calls` function (Figure 3), which compares average interval widths (and the IQR for interval widths) across scenarios based on the number of recordings validated. An example is provided below:

```
# obtain summary of validation effort as an object that can be used with
# the summary plot functions plot_x_vs_calls
s <-  summarize_n_validated(
  data_list = sim_data$masked_dfs,
  theta_scenario = "FE", # must not be NULL for the plot_X_vs_calls functions
  scenario_numbers = 1:4
)
```

```
plot_width_vs_calls(
  sim_summary = sims_output,
  calls_summary = s,
  regex_pars = "lambda",
  theta_scenario = "FE",
  scenarios = 2:5
) + # add horizontal line at target 95% CI width
  geom_hline(yintercept = 3, linetype = "dotted")
```

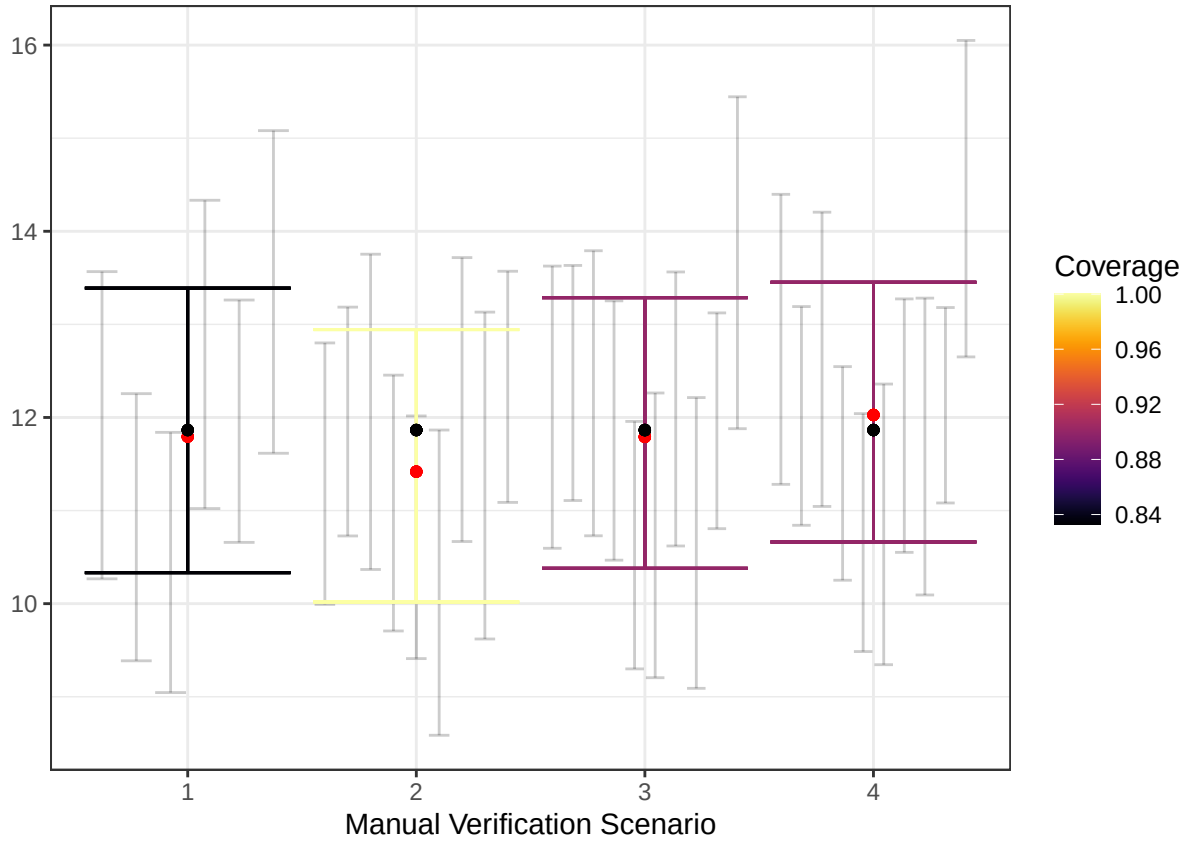


Figure 2: Example output from `visualize_single_parameter` is similar to that of `visualize_parameter_group`: the x-axis shows the validation scenario, and the y-axis shows the true value of the parameter (black point). Average posterior means are shown by red points, individual 95% posterior intervals from converged model fits are shown as small grey error bars and average 95% posterior intervals are shown by the larger colored error bars. The coverage is indicated by color.

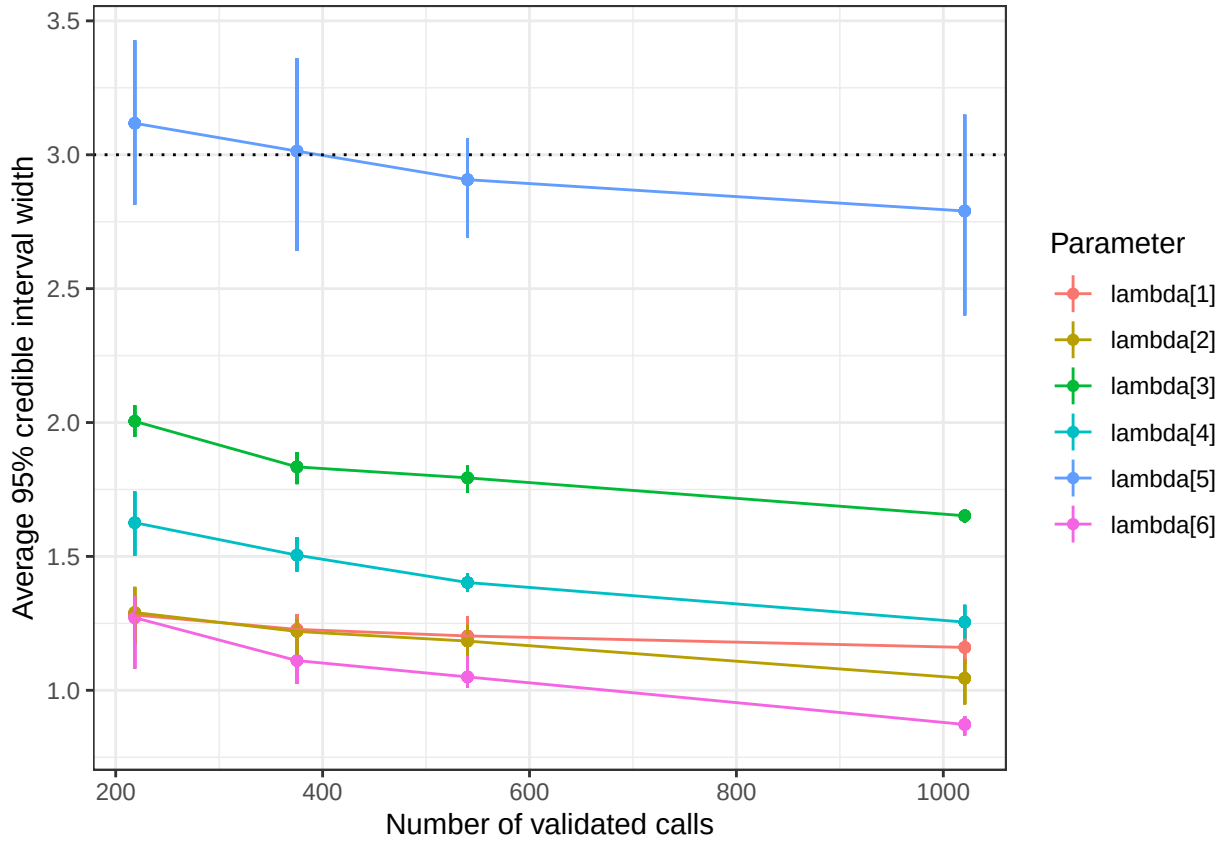


Figure 3: Example output from `plot_width_vs_calls`. The x-axis shows the number of validated recordings. Note that the level of effort increases with scenario number for the fixed-effort designs; this may not always be the case, for example, with a stratified-by-species design as in the main text. The y-axis shows the value of the average 95% posterior interval width (point). The middle 50% of interval widths for each parameter are shown by the error bars. Color indicates the parameter.

We provide analogous functions `plot_coverage_vs_calls` and `plot_bias_vs_calls` taking identical arguments that show how coverage and estimation error change with the number of calls validated. Note that in all of our visualization functions, if no fitted models have $\hat{R} \leq c$ for all parameters, where c is the specified convergence threshold under a given scenario, then the scenario will not appear on the x-axis. In our example, we used $c = 1.1$.

To ensure that there aren't substantial problems with estimation of other model parameters, we can create plots for ψ and Θ:

```
visualize_parameter_group(
  sim_summary = sims_output,
  pars = "psi",
  theta_scenario = 1,
  scenarios = 1:4,
  convergence_threshold = 1.05
)
```

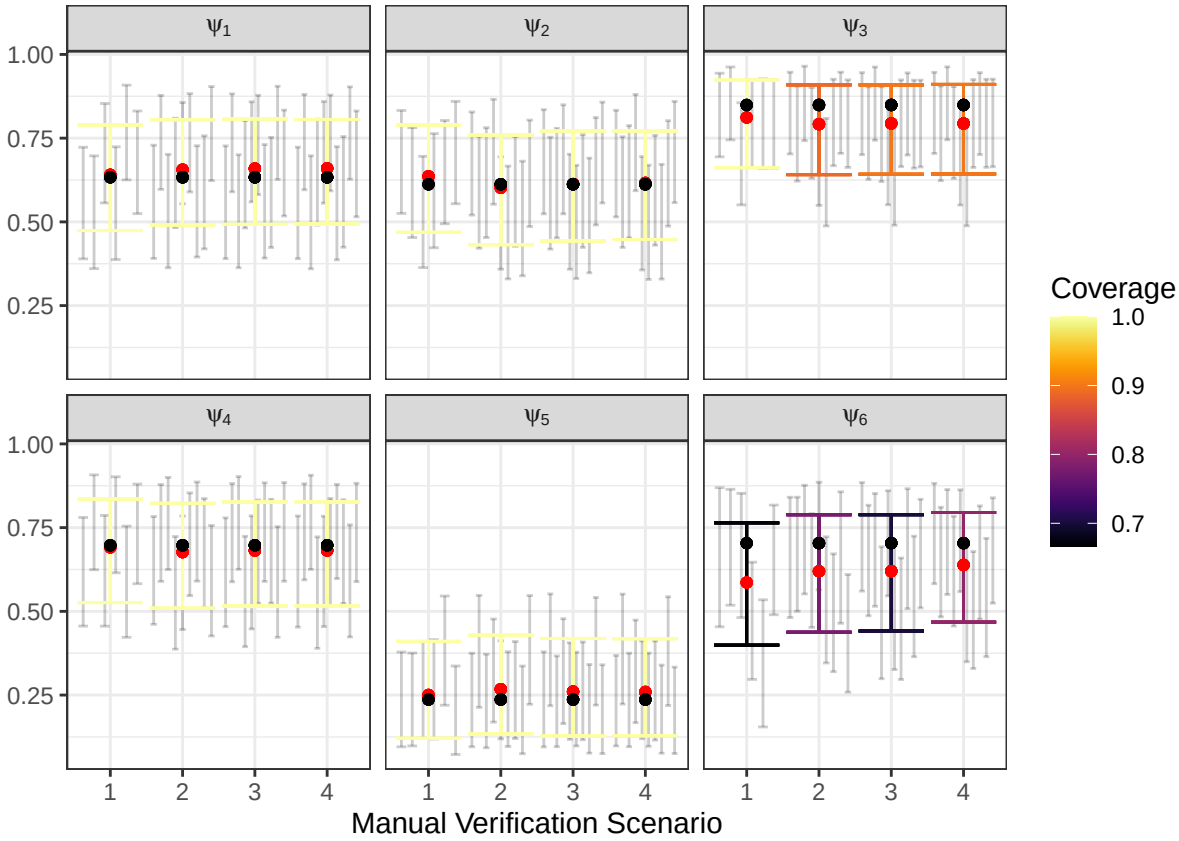


Figure 4: We encourage users to check inference for other model parameters, even if they are not explicitly of interest based on measurable objectives. Here, we examine $\boldsymbol{\psi}$. Note that coverage for several parameters is slightly low. For the purpose of exposition, we only fit 10 datasets and we expect that coverage would be near nominal levels if 50 or more datasets were used in simulation.

```r
visualize_parameter_group(
  sim_summary = sims_output,
  pars = "theta",
  theta_scenario = 1,
  scenarios = 1:4,
  convergence_threshold = 1.05
)
```

The simulation results shown in Figure 4 and 5 don't immediately cause concern that inference for relative activity rates will be severely compromised for any of these validation designs. Some parameters have slightly low coverage, but we believe this is mostly due to the small number of datasets used in this example; users should increase the number of datasets to at least 50, which we expect would lead to coverage near nominal levels.

2.8 Take-aways from the Fixed-Effort Simulations

Based on Figures 1 - 3, we can rule out validation scenario 1, because the expected width of 95% posterior intervals is greater than the threshold value of 3. Because of this, we might choose validation scenario 2 as one possible fixed effort validation design. This fixed effort design assumes that 10% of recordings from each site-night are randomly selected for validation, leading to around 375 recordings being validated per dataset. In our simulations, posterior means for each λ_k appear to be nearly unbiased, with near-nominal coverage and average posterior interval widths less than or equal to 3 for this scenario. However, it is possible that for a given dataset, the interval width for λ_5 will be larger than 3, regardless of the number of validated calls; error bars in Figure 3 overlap or exceed the dotted line in all scenarios.

In situations such as this one, a decision is required of the user. If a slightly larger interval width is acceptable, validating 10% percent of recordings from all site-visits is reasonable. If it is not, then the LOVE will need to be increased beyond that in scenario 4 (1021 recordings). The question is whether potentially tripling the number of validated recordings will be worth the benefit of ensuring the relative activity level for species 5 is less than 3. This is a question about measurable objectives that we leave to the practitioner and their study priorities.

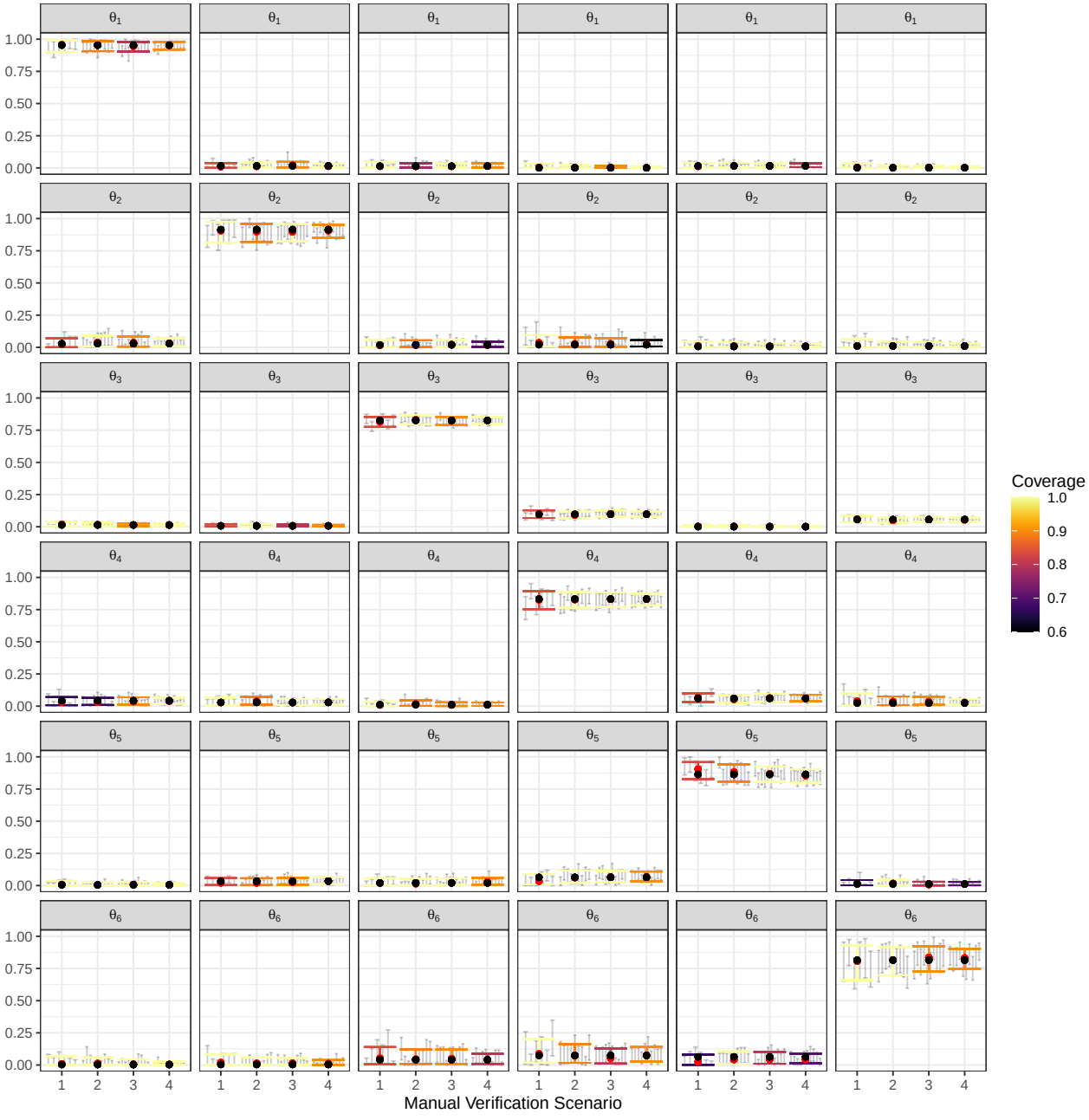


Figure 5: Checking inference for the elements of the classification matrix Θ. For most parameters (facets), validation scenario 2 appears to lead to minimal estimation error and near-nominal coverage.

3 Stratified-by-species example

In this section, we provide further details of the example in the main manuscript, which assumed a stratified-by-species design. In a stratified-by-species design, the random mechanism is a stratified random sample, with strata formed by the species labels assigned during automated classification (i.e., by autoID). In this type of design, the LOVE is the proportion of recordings selected to be validated for each species label.

We repeat the simulation study conducted in the main text in this section, providing greater detail for some of the steps. We assume the same measurable objectives and cost constraints as in Section 2.

3.1 Simulate data

Using the same values for `psi`, `lambda` and `Theta` as in Section 2.4, we simulate data using `simulate_validatedData` with `design_type = "BySpecies"`. This will yield simulated data in the format described in Section 2.4.

```
sim_data <- simulate_validatedData(
  n_datasets = 10,
  design_type = "BySpecies",
  scenarios = list(
    spp1 = 0.15,
    spp2 = 0.15,
    spp3 = c(0.15, .5, 1),
    spp4 = 0.1,
    spp5 = c(0.5, 1),
    spp6 = 0.25
  ),
  nsites = nsites,
  nspecies = nspecies,
  nvisits = nvisits,
  psi = psi,
  lambda = lambda,
  theta = Theta,
  directory = here::here("Vignette", "BySpecies")
)
```

Note that in contrast with Section 2.4, `simulate_validatedData` expects the `scenarios` argument to be a list of proportions corresponding to the possible levels of effort for each species when specifying a stratified-by-species design. Internally, `simulate_validatedData` calls `base::expand.grid`, and considers all possible combinations of the various levels of effort for each species, meaning that the number of simulated scenarios grows extremely quickly. The biggest implication of having a larger number of simulation scenarios to consider is increased computation time. For example, in the previous code block we fixed the level of effort

for species 1, 2, 4 and 6 at .15, .15, .1, and .25, respectively. There are three possible proportions to validate for species 3 and two possible levels of effort for species 5, yielding six possible scenarios, which are summarized in the additional output `sim_data$scenarios_df` that is available when `design_type = "BySpecies"` is specified:

```
names(sim_data)
```

```
## [1] "full_datasets" "zeros"         "masked_dfs"    "scenarios_df"
```

```
sim_data$scenarios_df
```

```
##   scenario spp1 spp2 spp3 spp4 spp5 spp6
## 1        1 0.15 0.15 0.15  0.1  0.5 0.25
## 2        2 0.15 0.15 0.50  0.1  0.5 0.25
## 3        3 0.15 0.15 1.00  0.1  0.5 0.25
## 4        4 0.15 0.15 0.15  0.1  1.0 0.25
## 5        5 0.15 0.15 0.50  0.1  1.0 0.25
## 6        6 0.15 0.15 1.00  0.1  1.0 0.25
```

If users would like to supply a tailored set of validation scenarios, this can be done through the `scen_expand` and `scen_df` arguments as follows:

```
# Define the scenarios dataframe. Each row is a scenario, each column is a spp
my_scenarios <- data.frame(
  spp1 = c(0.05, 0.05, 0.1),
  spp2 = c(0.1, 0.2, 0.3),
  spp3 = c(0.2, 0.2, 0.25),
  spp4 = c(0.4, 0.5, 0.6),
  spp5 = rep(1, 3),
  spp6 = rep(0.5, 3)
)

sim_data2 <- simulate_validatedData(
  n_datasets = 10,
  design_type = "BySpecies",
  scen_expand = FALSE,
  scen_df = my_scenarios,
  scenarios = NULL,
  nsites = nsites,
  nspecies = nspecies,
  nvisits = nvisits,
  psi = psi,
  lambda = lambda,
  theta = Theta,
  directory = here::here("Vignette", "BySpecies")
)

sim_data2$scenarios_df # the same as my_scenarios above
```

```
##   scenario spp1 spp2 spp3 spp4 spp5 spp6
## 1        1 0.05  0.1 0.20  0.4    1  0.5
## 2        2 0.05  0.2 0.20  0.5    1  0.5
## 3        3 0.10  0.3 0.25  0.6    1  0.5
```

In the remainder of this example, we use the `sim_data` object. We can combine the `scenarios_df` output with the output of `summarize_n_validated` to understand what the scenarios are and how many recordings are validated under each.

```
summary1 <- sim_data$scenarios_df
summary2 <- summarize_n_validated(
  sim_data$masked_dfs,
  scenario_numbers = 1:6,
  theta_scenario = "BySpecies"
)

call_sum <- left_join(summary1, summary2, by = "scenario")

call_sum
```

```
##   scenario spp1 spp2 spp3 spp4 spp5 spp6 theta_scenario n_validated
## 1        1 0.15 0.15 0.15  0.1  0.5 0.25      BySpecies       603.8
## 2        2 0.15 0.15 0.50  0.1  0.5 0.25      BySpecies      1018.3
## 3        3 0.15 0.15 1.00  0.1  0.5 0.25      BySpecies      1610.4
## 4        4 0.15 0.15 0.15  0.1  1.0 0.25      BySpecies       778.3
## 5        5 0.15 0.15 0.50  0.1  1.0 0.25      BySpecies      1192.8
## 6        6 0.15 0.15 1.00  0.1  1.0 0.25      BySpecies      1784.9
```

In the resulting dataframe, we see that scenario 1 has the lowest overall level of validation effort: species 1-3 have 15% of their recordings validated, species 4 has 10% validated, species 5 has 50%, and species 6 has 25%, yielding around 604 recordings validated in an average dataset. We use one dataset simulated under scenario 1 to tune the MCMC.

3.2 Tune the MCMC

The MCMC tuning in this step is very similar to the procedures used in Section 2.5. We begin by fitting a model to one dataset using the `tune_mcmc` function.

```
i <- sample(1:length(sim_data$full_datasets), 1)

tune_list <- tune_mcmc(
  dataset = sim_data$masked_dfs[[1]][[i]],
  zeros = sim_data$zeros[[i]]
)
```

```
## [1] "Fitting MCMC in parallel ... this may take a few minutes"
```

As in Section 2.5, we visually inspect the chains for convergence using trace plots. We set trace plot windows to be wider than the suggested values from `tune_mcmc`, meaning a lower starting value and a higher ending value. Based on these plots, chains appear to be mixing well as no one chain stands out visually.

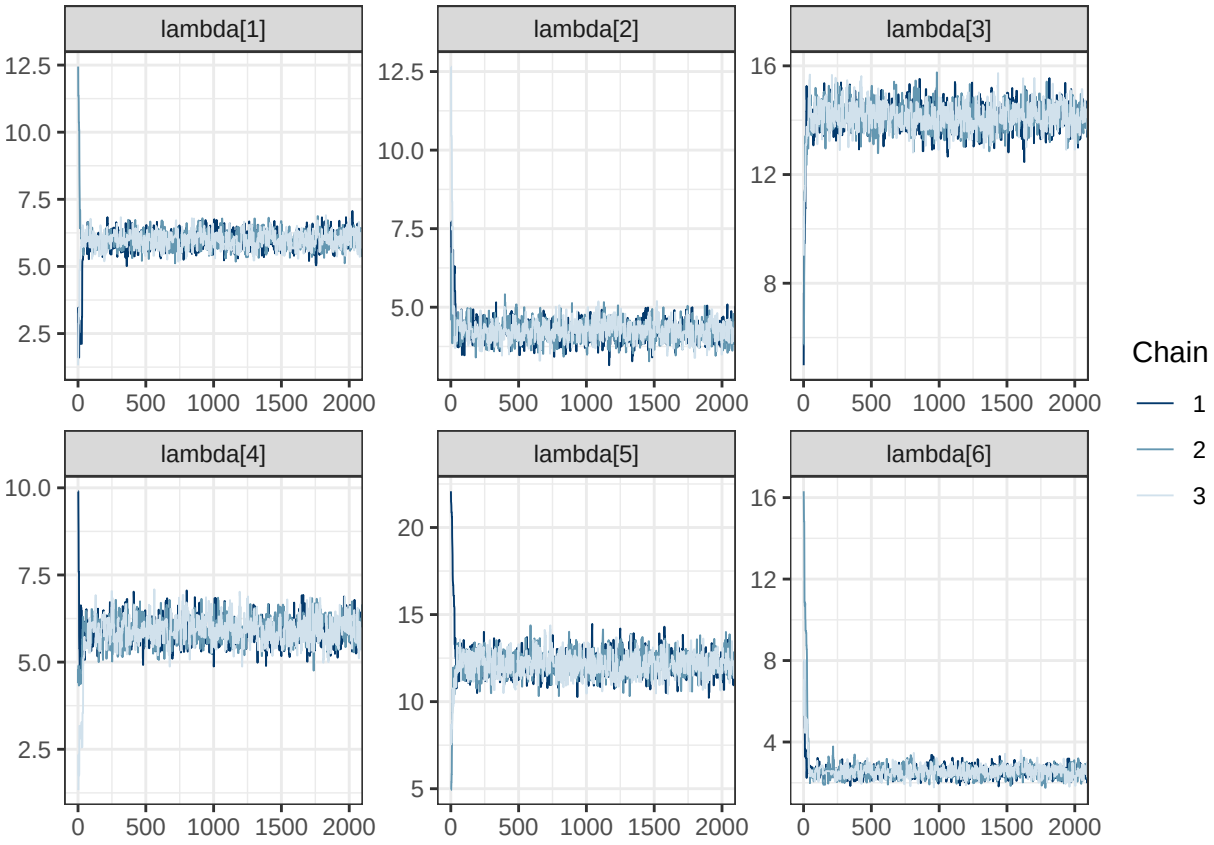
```
tune_list$min_warmup
```

```
## [1] 500
```

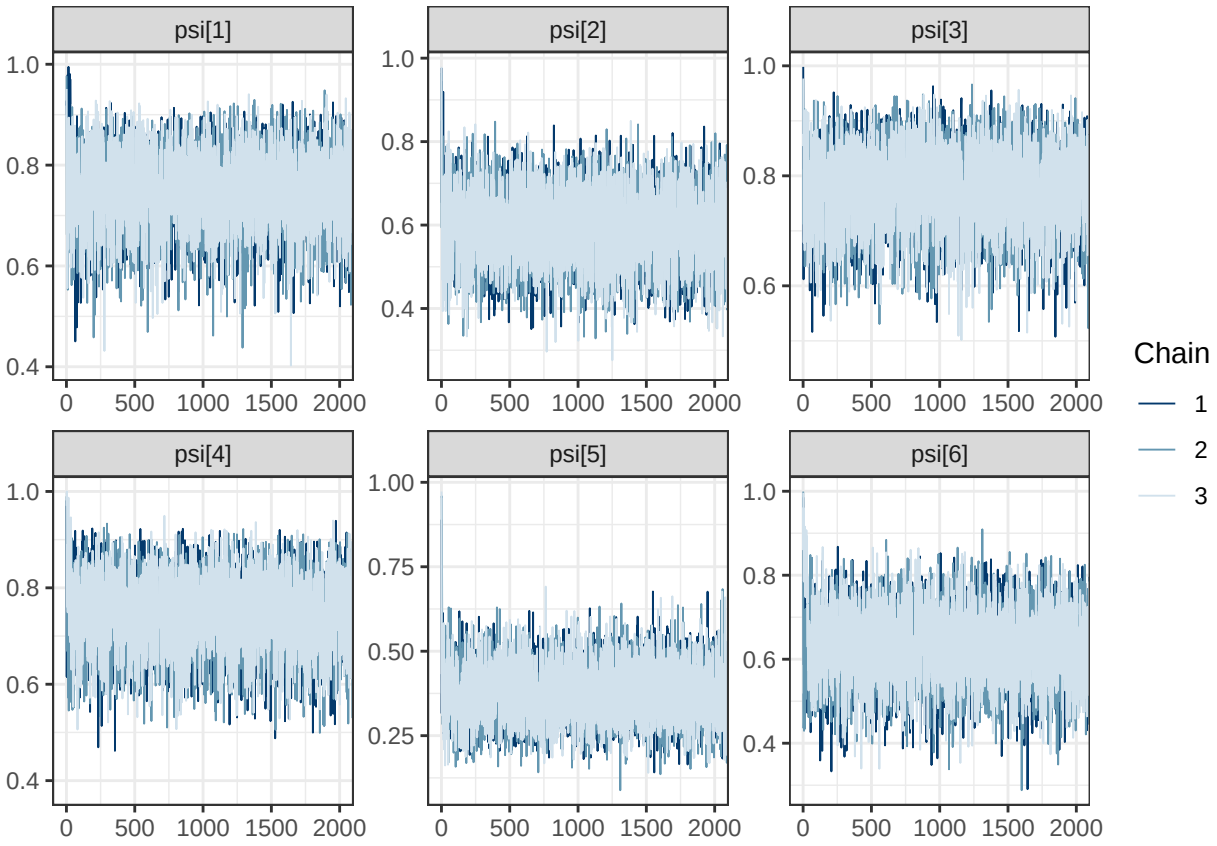
```
tune_list$min_iter
```

```
## [1] 1500
```

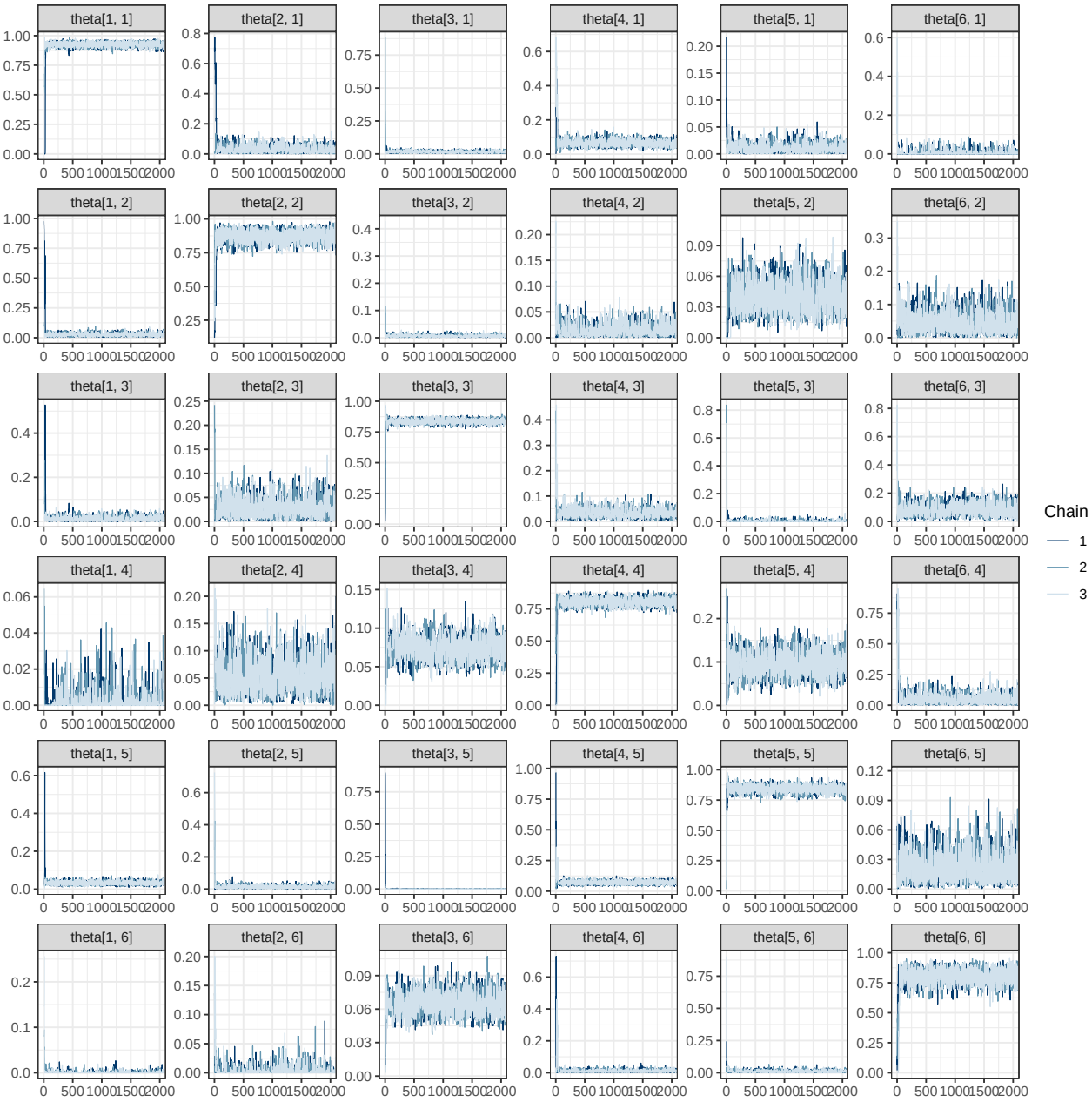
```
# increase iters to visualize beyond the warmup and
# iterations output from tune_mcmc
fit <- tune_list$fit

mcmc_trace(fit, regex_pars = "lambda", window = c(0, tune_list$min_iter + 500))
```



```
mcmc_trace(fit, regex_pars = "psi", window = c(0, tune_list$min_iter + 500))
```




```
mcmc_trace(fit, regex_pars = "theta", window = c(0, tune_list$min_iter + 500))
```



In all trace plots, it appears that MCMC chains have reached approximate convergence well before the warmup value of 500 draws output from `tune_mcmc`.

Next, we examine the effective sample sizes in the tails and bulk of the posteriors and the $\hat{R}$ values:

```
tune_list$MCMC_diagnostics
```

```
##       parameter  ess_bulk  ess_tail      Rhat
## 1    lambda[1]  4586.982  5993.003 1.0008212
## 2    lambda[2]  3581.838  4914.658 1.0010488
## 3    lambda[3]  4175.436  5574.614 1.0003131
```

```
## 4    lambda[4]  2900.041  4380.094 1.0016083
## 5    lambda[5]  3802.321  5259.429 1.0007258
## 6    lambda[6]  3101.917  3661.574 1.0013299
## 7       psi[1] 29776.047 28831.669 1.0000620
## 8       psi[2] 29576.345 27700.001 1.0000145
## 9       psi[3] 29556.265 29705.613 1.0001375
## 10      psi[4] 29032.052 29084.401 1.0000431
## 11      psi[5] 29296.485 29557.225 0.9999959
## 12      psi[6] 21581.199 20365.686 1.0000135
## 13 theta[1, 1]  5692.854 10453.234 1.0007769
## 14 theta[2, 1]  3909.942  5105.449 1.0004555
## 15 theta[3, 1]  4434.432  5738.438 1.0002853
## 16 theta[4, 1]  5595.319  7893.852 1.0001619
## 17 theta[5, 1]  4628.803  5215.027 1.0003833
## 18 theta[6, 1]  2691.909  1873.632 1.0029418
## 19 theta[1, 2]  4151.707  6452.476 1.0025177
## 20 theta[2, 2]  4668.185  7425.430 1.0009754
## 21 theta[3, 2]  4838.225  6110.314 1.0008558
## 22 theta[4, 2]  3500.227  4971.066 1.0032900
## 23 theta[5, 2]  4967.287  4079.468 1.0003252
## 24 theta[6, 2]  3976.682  4010.090 1.0006046
## 25 theta[1, 3]  4583.505  5090.999 1.0005850
## 26 theta[2, 3]  4284.615  4711.188 1.0002337
## 27 theta[3, 3]  5105.161  9654.830 1.0002040
## 28 theta[4, 3]  4408.827  5921.990 1.0006261
## 29 theta[5, 3]  2797.878  3211.883 1.0001722
## 30 theta[6, 3]  4115.929  5059.948 1.0010917
## 31 theta[1, 4]  3004.171  2900.188 1.0003341
## 32 theta[2, 4]  3410.903  4958.206 1.0011579
## 33 theta[3, 4]  3719.559  6909.601 1.0010432
## 34 theta[4, 4]  5990.556  8347.173 1.0006233
## 35 theta[5, 4]  4278.991  5809.358 1.0017183
## 36 theta[6, 4]  3861.842  5279.086 1.0013809
## 37 theta[1, 5]  6801.137  9982.209 1.0001797
## 38 theta[2, 5]  5821.074  6378.238 1.0008976
## 39 theta[3, 5]  3480.448  3118.654 1.0006837
## 40 theta[4, 5]  6963.758  8876.021 1.0008540
## 41 theta[5, 5]  5041.023  8564.590 1.0010372
## 42 theta[6, 5]  6060.001  6357.984 1.0005745
## 43 theta[1, 6]  3889.227  4243.724 1.0016516
## 44 theta[2, 6]  3461.016  3869.012 1.0004007
## 45 theta[3, 6]  5593.781  7652.210 1.0002469
## 46 theta[4, 6]  5167.258  6050.292 1.0004419
## 47 theta[5, 6]  4038.106  4580.835 1.0005979
## 48 theta[6, 6]  4083.018  5745.339 1.0013112
```

For all parameters, the bulk and tail effective sample sizes are fairly large: if we slightly decrease the number of post-warmup draws, we could expect to characterize both the center and tails of the posteriors well. Additionally, $\hat{R} \approx 1.00$ for all parameters, indicating good mixing for chains. As before, we check this suspicion by computing MCMC diagnostic statistics for truncated chains:

```r
# for each chain, extract iterations 501:2000 for all parameters
shortened <- lapply(fit, function(x) x[(tune_list$min_warmup):tune_list$min_iter + 1000,])

# summarize the shortened chains and select the effective sample
# size columns
mcmc_sum(shortened, truth = rep(0, ncol(shortened[[1]]))) %>%
  select(parameter, ess_bulk, ess_tail)
```

```
##      parameter  ess_bulk  ess_tail
## 1    lambda[1]  472.0693  746.2314
## 2    lambda[2]  462.5433  699.5027
## 3    lambda[3]  471.1166  533.4401
## 4    lambda[4]  397.2467  652.5985
## 5    lambda[5]  368.5811  436.7193
## 6    lambda[6]  348.8244  419.3278
## 7       psi[1] 2887.6340 2774.5575
## 8       psi[2] 2767.6607 2682.1420
## 9       psi[3] 2783.6565 2347.5190
## 10      psi[4] 2852.2354 2860.5767
## 11      psi[5] 2871.2444 2799.4130
## 12      psi[6] 2403.2472 3022.6069
## 13 theta[1, 1]  661.0220 1042.3652
## 14 theta[2, 1]  430.3098  459.8766
## 15 theta[3, 1]  420.1689  659.9489
## 16 theta[4, 1]  595.7169  880.2612
## 17 theta[5, 1]  407.5506  504.8551
## 18 theta[6, 1]  286.5996  167.0759
## 19 theta[1, 2]  428.5388  642.9276
## 20 theta[2, 2]  643.0888  960.3424
## 21 theta[3, 2]  465.2884  652.4888
## 22 theta[4, 2]  434.5653  480.1366
## 23 theta[5, 2]  574.7982  817.3303
## 24 theta[6, 2]  425.6721  360.8348
## 25 theta[1, 3]  562.5931  589.5009
## 26 theta[2, 3]  445.7846  344.7673
## 27 theta[3, 3]  602.9921 1030.5888
## 28 theta[4, 3]  442.7124  686.7902
## 29 theta[5, 3]  421.9302  457.0277
## 30 theta[6, 3]  454.4696  550.7088
## 31 theta[1, 4]  344.6359  385.4225
## 32 theta[2, 4]  352.3140  479.0098
## 33 theta[3, 4]  353.5589  647.7829
## 34 theta[4, 4]  628.6587 1218.6964
## 35 theta[5, 4]  592.6895  948.9038
## 36 theta[6, 4]  421.4482  649.1380
## 37 theta[1, 5]  869.1375  843.5989
## 38 theta[2, 5]  592.9222  599.7965
## 39 theta[3, 5]  366.5789  420.8605
## 40 theta[4, 5]  667.1658  892.2651
## 41 theta[5, 5]  549.1402 1104.2461
## 42 theta[6, 5]  536.2736  553.4037
## 43 theta[1, 6]  387.0690  563.8451
## 44 theta[2, 6]  344.3959  291.9980
```

```
## 45 theta[3, 6]  617.9373  825.0112
## 46 theta[4, 6]  642.9937  609.9640
## 47 theta[5, 6]  571.2368  749.5565
## 48 theta[6, 6]  564.3976  970.1239
```

The results appear satisfactory, with effective sample sizes of more than 250 in both the tail and bulk of the posterior distributions for each parameter. Based on the results of MCMC exploration, it appears that using an MCMC with 2500 iterations with 500 discarded as warmup is likely to produce good results for our simulation study.

3.3 Fit models

```
sims_out <- run_sims(
  data_list = sim_data$masked_dfs,
  zeros_list = sim_data$zeros,
  DGVs = list(lambda = lambda, psi = psi, theta = Theta),
  theta_scenario_id = "BySpecies",
  parallel = TRUE,
  niter = tune_list$min_iter + 1000,
  nburn = tune_list$min_warmup + 500,
  thin = 1,
  save_fits = FALSE,
  save_individual_summaries_list = FALSE,
  directory = here::here("Vignette", "BySpecies")
)
```

```
## Beginning scenario 1.

## 2025-02-12 12:24:28.510301

##   |                                                                      |

## Beginning scenario 2.

## 2025-02-12 12:36:43.350071

##   |                                                                      |

## Beginning scenario 3.

## 2025-02-12 12:49:19.755398

##   |                                                                      |

## Beginning scenario 4.
```

```
## 2025-02-12 12:59:41.090194

##   |                                                                      |

## Beginning scenario 5.

## 2025-02-12 13:08:22.333299

##   |                                                                      |

## Beginning scenario 6.

## 2025-02-12 13:17:43.482075

##   |                                                                      |
```

The output, `sims_out`, will have the same structure as described in Section 2.6.

3.4 Visualize results

Recall that the measurable objectives of our study are to estimate the relative activity rates with estimation error less than 1 call per night and a 95% posterior interval width of less than 3 calls per night. Furthermore, we assume that the monitoring program can afford to validate 4000 calls in total. All of the possible validation designs shown in `sim_data$scenarios_df` are feasible given this budget.

We begin with detailed plots of relative activity rates, occurrence probabilities, and classification probabilities.

```
visualize_parameter_group(
  sim_summary = sims_out,
  pars = "lambda",
  theta_scenario = "BySpecies",
  scenarios = 1:6,
)
```

Based on the simulation results shown in Figure 6, any of validation scenarios 1-6 is expected to produce a posterior mean estimate for each relative activity parameter that is near the true value. All models converged for all scenarios. Coverage varies by scenario for each parameter, but recall that we only fit models to 10 datasets.

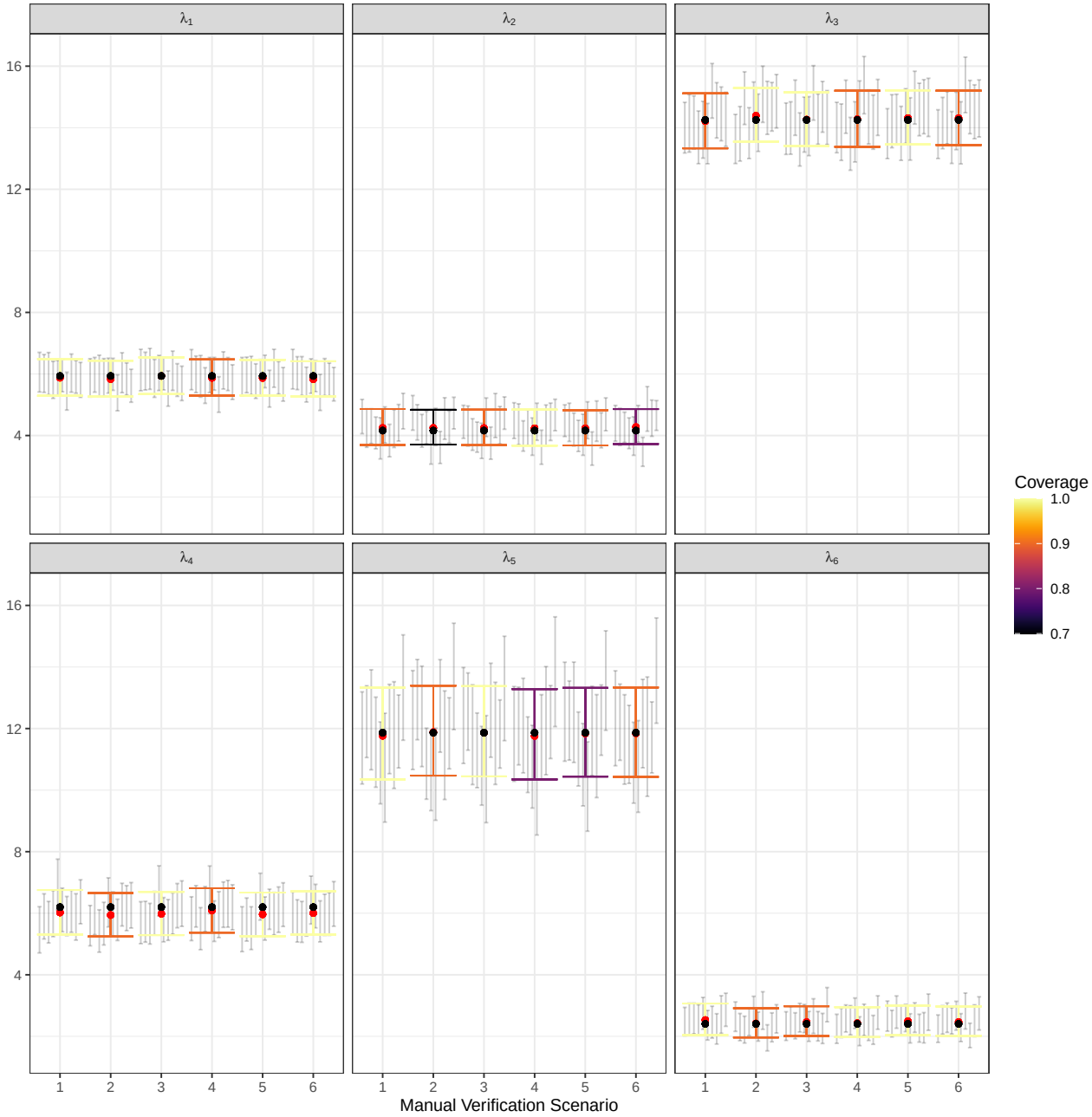


Figure 6: Output from `visualize_parameter_group` under six possible stratified-by-species scenarios for relative activity rates. Parameters are shown in each facet and validation scenario number is on the x-axis. Small grey intervals are 95% posterior intervals for each parameter from fitted models that converged. Larger colored error bars are average 95% posterior intervals with the color indicating the coverage. Black dots are the true parameter value and red dots are average posterior means.

```
visualize_parameter_group(
  sim_summary = sims_out,
  pars = "psi",
  theta_scenario = "BySpecies",
  scenarios = 1:6,
)
```

The simulation results for occurrence probabilities show a small amount estimation error for all occurrence probabilities, with the size and direction varying depending on the species. For species with larger estimation error, coverage is also slightly low. However, since we are most concerned with relative activity, the goal is to check results for each ψ_k for severe estimation error and/or lack of coverage, which are not shown in Figure 7. Similarly, we do not observe alarming results in Figure 8.

```
visualize_parameter_group(
  sim_summary = sims_out,
  pars = "theta",
  theta_scenario = "BySpecies",
  scenarios = 1:6,
)
```

One measurable objective is for 95% posterior interval widths to be less than 3 calls per night for each species' relative activity rate. As in Section 2.7, the `plot_width_vs_calls` function provides Figure 9, which addresses this measurable objective directly.

```
plot_width_vs_calls(
  sim_summary = sims_out,
  calls_summary = call_sum,
  regex_pars = "lambda",
  theta_scenario = unique(sims_out$theta_scenario),
  scenarios = 1:6
)
```

Once again as in Section 2.7, the widest posterior interval width is for species 5. All validation scenarios have average 95% posterior interval widths near or slightly below 3 calls per night, but the error bars indicate an interval width greater than 3 is possible for any validation scenario. Scenario 6, in which 1785 recordings are validated, has the narrowest expected interval width.

3.5 Take-aways

The results in Section 3.4, imply that, of the stratified-by-species validation designs considered, scenario 6 offered the best results with respect to the measurable objectives we outlined in Section 2.2. In this scenario,

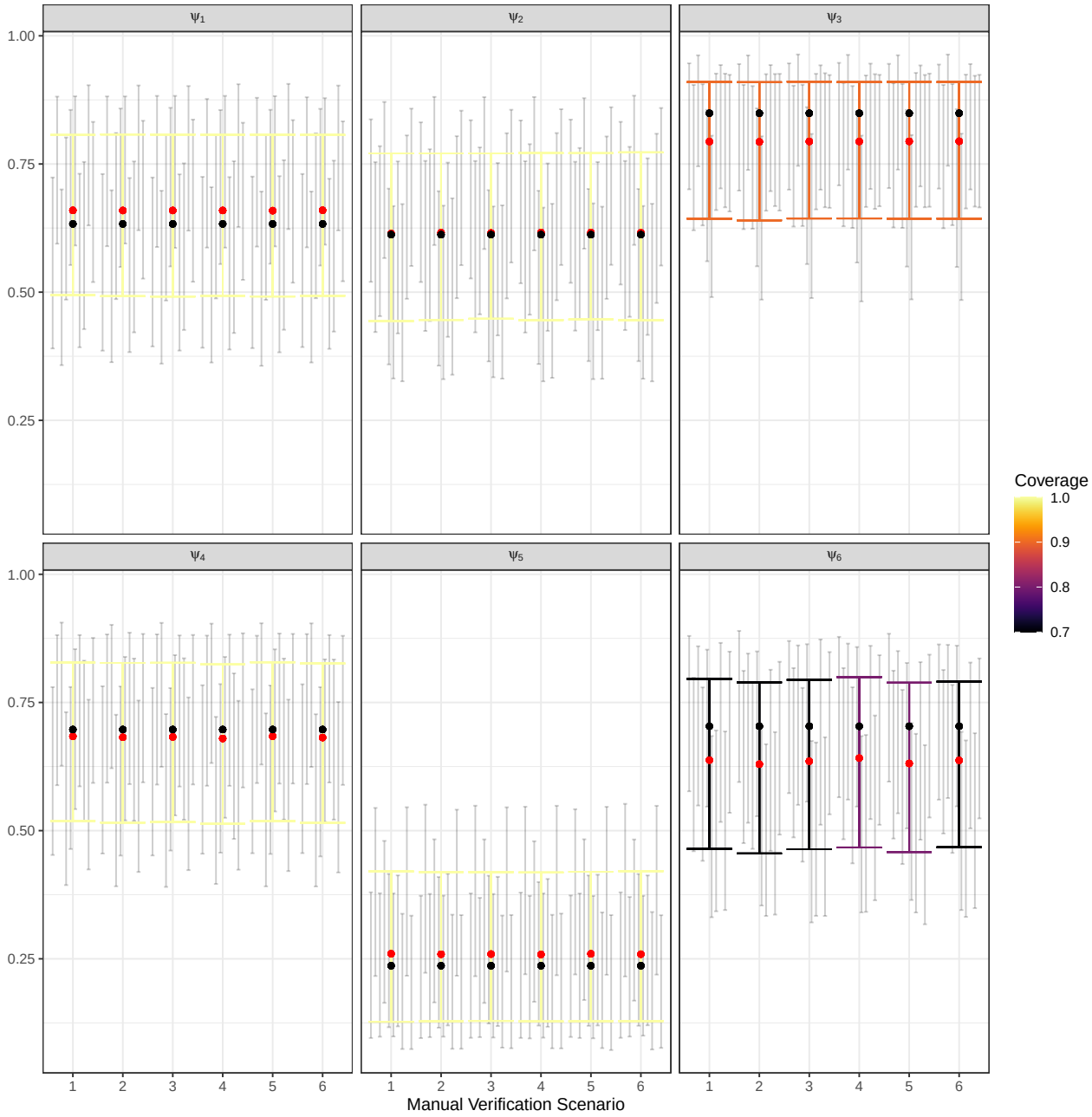


Figure 7: Output from `visualize_parameter_group` under six possible stratified-by-species scenarios for occurrence probabilities. Parameters are shown in each facet and validation scenario number is on the x-axis. Small grey intervals are 95% posterior intervals for each parameter from fitted models that converged. Larger colored error bars are average 95% posterior intervals with the color indicating the coverage. Black dots are the true parameter value and red dots are average posterior means.

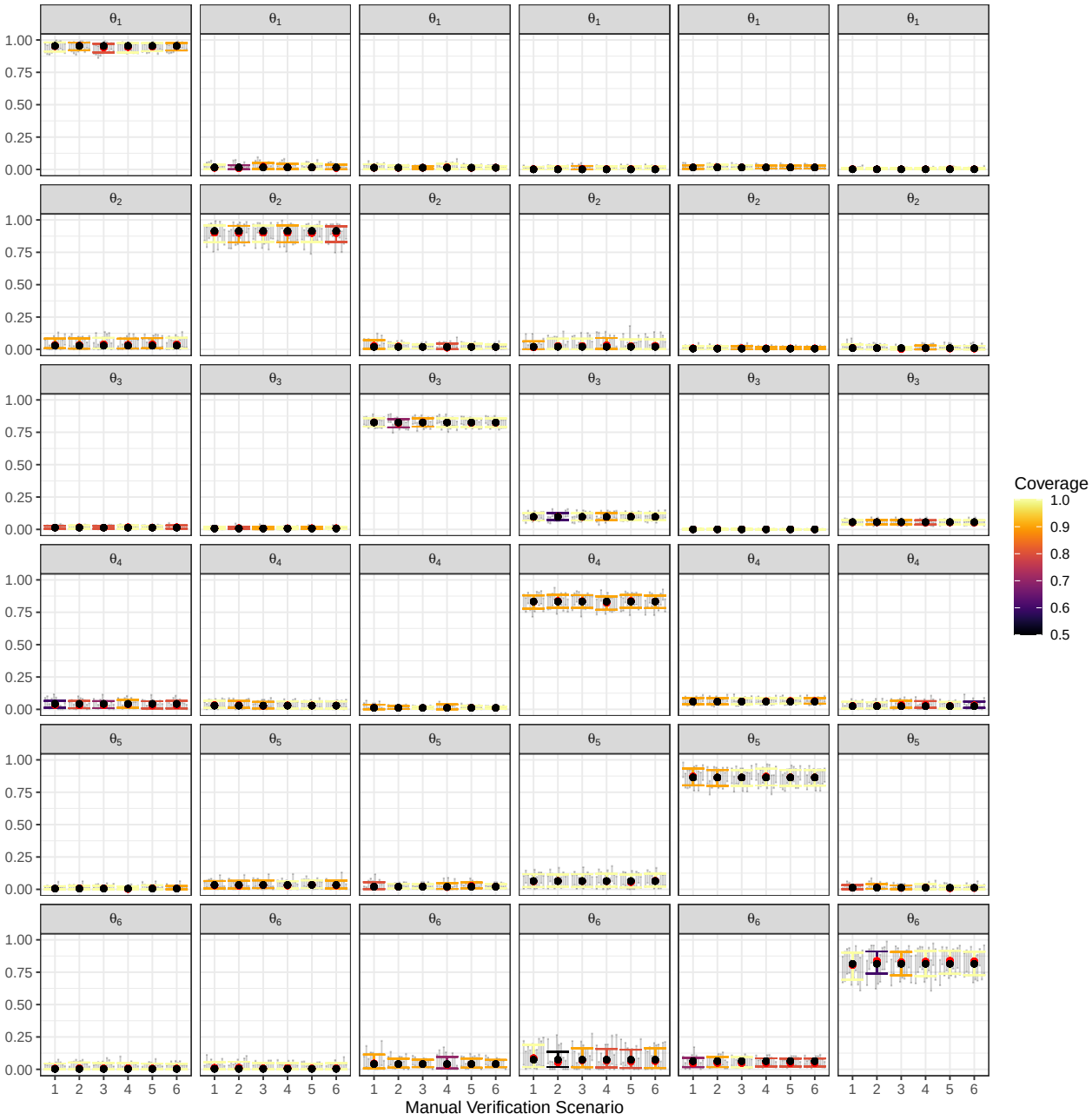


Figure 8: Output from `visualize_parameter_group` under six possible stratified-by-species scenarios for classification probabilities. Parameters are shown in each facet and validation scenario number is on the x-axis. Small grey intervals are 95% posterior intervals for each parameter from fitted models that converged. Larger colored error bars are average 95% posterior intervals with the color indicating the coverage. Black dots are the true parameter value and red dots are average posterior means.

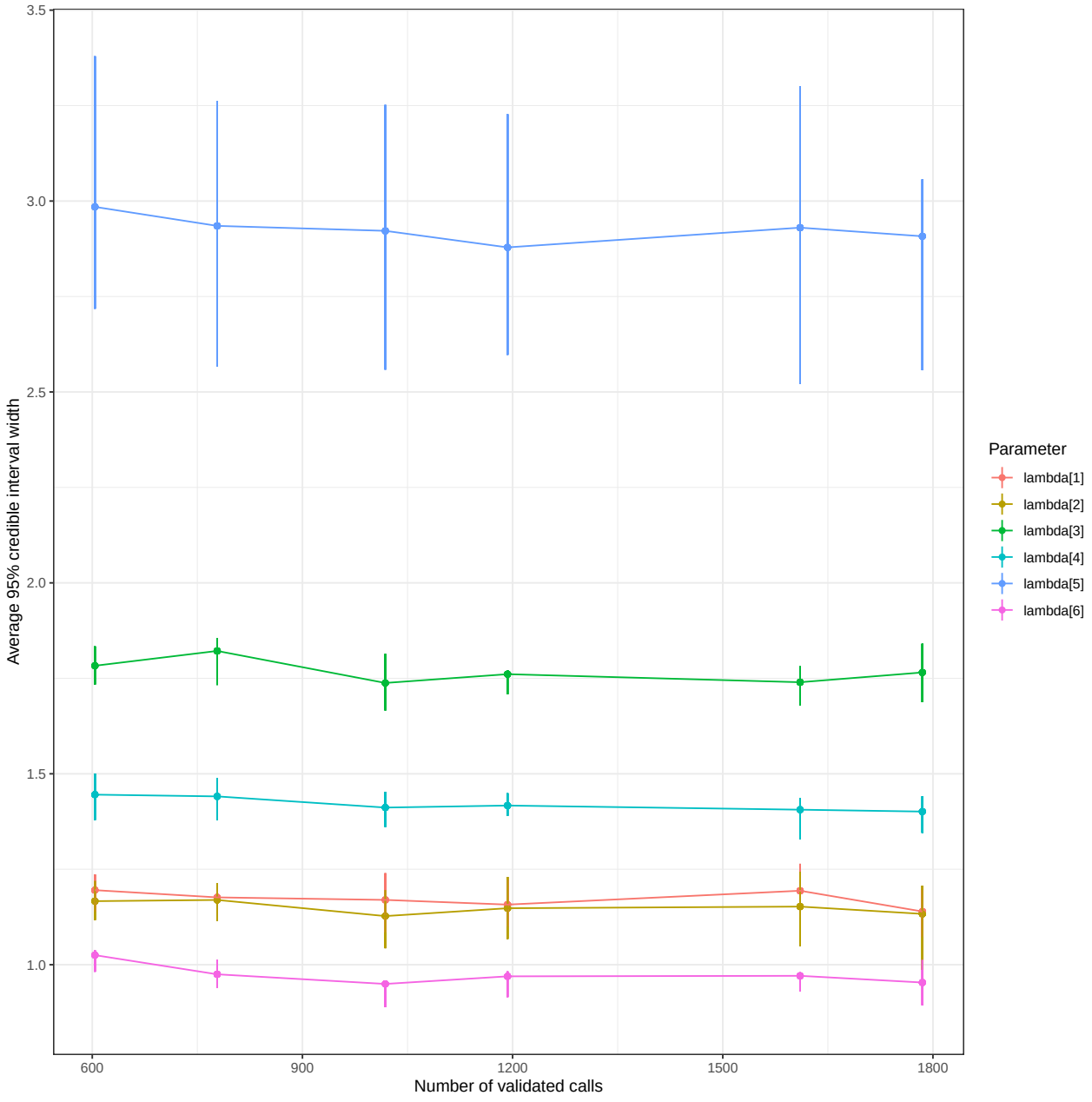


Figure 9: Plots of 95% posterior interval width vs the number of calls validated.

the six species in our assemblage receive 15%, 15%, 100%, 10%, 100%, and 25% of their recordings validated, respectively.

To keep the number of scenarios in this example small, we fixed the LOVE for several species. However, if the measurable objectives specified the desired accuracy and precision for estimates of occurrence probability, it may be desirable to consider alternative validation scenarios. For example, we consistently underestimated the occurrence probability for species 6 (Figure 7), and additional simulations under validation designs that varied the level of effort for species 6 would be warranted if measurable objectives related to occurrence probability for this species.

4 Conclusion

We have demonstrated the use of the `ValidationExplorer` package when the objective is to compare the merits of four competing LOVEs for fixed-effort and stratified-by-species designs. Specifically, in the fixed-effort design we considered validating a random sample of 5%, 10%, 15% or 30% of the recordings obtained during each visit to each site. In the stratified-by-species designs, we considered a suite of validation designs, fixing effort for species 1,2, 4, and 6 at less than or equal to 25%, and varied the level of effort for species 3 and 5. The exact simulations considered are summarized in the `call_sum` object in Section 3.1. The measurable objectives outlined in Section 2.2 were to estimate the relative activity parameters for each species, with near-nominal coverage and width less than 3 calls per night for 95% posterior intervals. Results, and their implications for these measurable objectives were summarized in Sections 2.8 and 3.5. Practitioners who would like to inform study design prior to data collection can repeat multiple simulation studies to see how inferences change with the number of sites and balanced visits, in addition to the validation design.

5 Table of Functions

Function	Argument	Description
simulate_validatedData	n_datasets	The number of datasets to be simulated
	design_type	The type of validation design. Must be one of "BySpecies" or "FixedPercent"
	scenarios	The possible levels of effort. If design_type = "BySpecies", this is provided as a list with vector-valued entries containing the possible percentages to validate for each species. If design_type = "FixedPercent", this argument is a vector of possible percentages.
	n_sites, n_visits, n_species	The number of sites, visits and species in the assemblage
	psi, lambda, theta,	Vectors of length nspecies containing the the parameter values for each species
	directory	The working directory where datasets are to be saved if the following arguments are set to TRUE
	save_datasets, save_masked_datasets	Logicals indicating whether to save each type of dataset
summarize_n_validated	data_list	A list of simulated (masked) datasets output from simulate_validatedData
	zeros_list	A list of true species/autoID combinations that were never observed at each site-visit.
	theta_scenario	An optional character string identifying the classifier scenario. If output from summarize_n_validated is to be used with any of the plot_X_vs_calls functions described below, this string must match the theta_scenario_id argument supplied to run_sims.
run_sims	data_list	A nested list of masked datasets in the format output from simualate_validatedData. The first layer of the list corresponds to scenarios, with each entry containing a list of n_datasets validated according to the scenario.

	`zeros_list`	A list of length `n_datasets` containing the site-visit-true-autoID combinations that were never observed.
	`DGVs`	A named list of the true data-generating values with entries `"psi", "lambda"` and `"theta"`.
	`theta_scenario_id`	An ID to show which classifier the scenario is under. This is provided as a convenience for the user if multiple simulation studies are to be conducted.
	`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.
	`n_iter, nburn, thin`	The number of iterations, warmup and thinning interval for each chain in the MCMC. Default values are 2000, 1000, and 1, respectively.
	`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. Default value is FALSE.
	`directory`	Where to save fits and summaries. Required if `save_fits = TRUE` or `save_individual_summaries_list = TRUE`. Default value is the current working directory given by `here::here()`
`visualize_parameter_group`	`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.
`visualize_single_parameter`		Arguments are identical to `visualize_parameter_group`
`plot_bias_vs_calls`, `plot_coverage_vs_calls`, `plot_width_vs_calls`	`sims_summary`	The summary output in the format from `run_sims`
	`calls_summary`	A summary of the number of calls validated per scenario. Expected format is that of output from `summarize_n_validated`.
	`pars`	The parameters to visualize.
	`regex_pars`	A group of parameters to visualize. One of `"psi"`, `"lambda"` or `"theta"`.
	`theta_scenario`	The classifier scenario ID.
	`scenarios`	The scenarios to be compared.
	`convergence_threshold`	At what value is an MCMC algorithm considered "converged"?

Table 2: Argument descriptions for each function in the `ValidationExplorer` software. My hope is that in the eventual final paper, this will be a scrollable html table

References

Gabry, Jonah, Daniel Simpson, Aki Vehtari, Michael Betancourt, and Andrew Gelman. 2019. "Visualization in Bayesian Workflow." *J. R. Stat. Soc. A* 182: 389–402. https://doi.org/10.1111/rssa.12378.

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.