# Vignette: How to simulate and fit data with stratified-by-species sampling for validation

Jacob Oram

2024-02-13

## Overview

This document accompanies the manuscript "Investigation into stratified-by-species validation of species labels for acoustic surveys", so that the reader may conduct similar simulations on their own. The manuscript provides a complete example analysis; here, we focus on the mechanics of setting up and running the necessary code to recreate the simulation study and explore alternative validation designs for different species assemblages.

### Background

The simulations we describe here and in the main manuscript are designed to explore the implications of using a stratified-by-species design to validate acoustic recordings that have already been classified by automated software algorithms. This simulation assumes an individual-level version of the count-detection model (as described in Section 2 of the main manuscript). Validation designs are determined by the proportion of calls with a given autoID label that are manually reviewed by human experts to determine the true species labels. These true species labels are assumed to be error free. The goal of the simulations described here is to explore how the level of effort assigned to each species may change posterior inference for a species assemblage of interest. The inputs for the simulations are the species assemblage, the classifier scenario, and the validation scenarios. The outputs are a summary dataframe for all datasets under all classifiers and scenarios; optionally, you may also save the raw summaries and draws from the fitted model. We describe how to set up these simulations below.

## Conducting your own simulation study

### 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)
```

### 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`.

### 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_BySpeciesValidation` function. Load this function by running

```
source("Data Simulation/simulate_BySpeciesValidation.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.

```
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 $i, j^{\text{th}}$ entry of the matrix (denoted $\theta_{ij}$) 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:

```
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:

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

```
##            [,1]       [,2]       [,3]
## [1,] 0.83929045 0.12708411 0.03362544
## [2,] 0.12703741 0.77818243 0.09478016
## [3,] 0.02015793 0.06666273 0.91317934
```

However you choose to generate the $\Theta$ matrix, make sure that the rows sum to 1:

```
rowSums(test_theta1)
```

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

```
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:

```
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
```

```
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 $2\times$ `n_datasets` + `n_datasets` $\times$ `nrow(scenarios_dataframe)` objects saved in your directory!

```
fake_data <- simulate_BySpeciesValidation(
  n_datasets = 5,
  scenarios_dataframe = 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 0.778     1     1    10
## 2     1     1        3      3      4   0.9 0.913     1     9    10
## 3     1     1        3      3      4   0.9 0.913     1     9    10
## 4     1     1        3      3      4   0.9 0.913     1     9    10
## 5     1     1        3      3      4   0.9 0.913     1     9    10
## 6     1     1        3      3      4   0.9 0.913     1     9    10
```

```
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 0.839      0     0    10
## 2     1     1        2      1      2   0.6 0.127      1     0    10
## 3     1     1        3      1      4   0.9 0.0202     1     0    10
## 4     1     1        1      2     11   0.3 0.127      0     0    10
## 5     1     1        3      2      4   0.9 0.0667     1     0    10
## 6     1     1        1      3     11   0.3 0.0336     0     0    10
```

```
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 0.913     1    10    10     1
## 2     1     1        3      3      4   0.9 0.913     1    10    10     2
## 3     1     1        3      3      4   0.9 0.913     1    10    10     3
## 4     1     1       NA      3      4   0.9 0.913     1    10    10     4
## 5     1     1        3      3      4   0.9 0.913     1    10    10     5
## 6     1     1        3      3      4   0.9 0.913     1    10    10     6
## # i 1 more variable: scenario <int>
```

**Step 4: Fit the data**

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_BySpeciesValidation`.
- `zeros_list`: The list of site-visit-true-autoID combinations that were never observed. This is the `zeros` object output from `simulate_BySpeciesValidation`.
- `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 $\lambda_k$ 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:

```
source("Model Fitting & Simulation/run_sims.R")
# takes about 2-4 minutes
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 = 10000, thin = 5,
         save_fits = FALSE,
         save_individual_summaries_list = FALSE,
         directory = paste0(here::here(), "/Testing"))
```

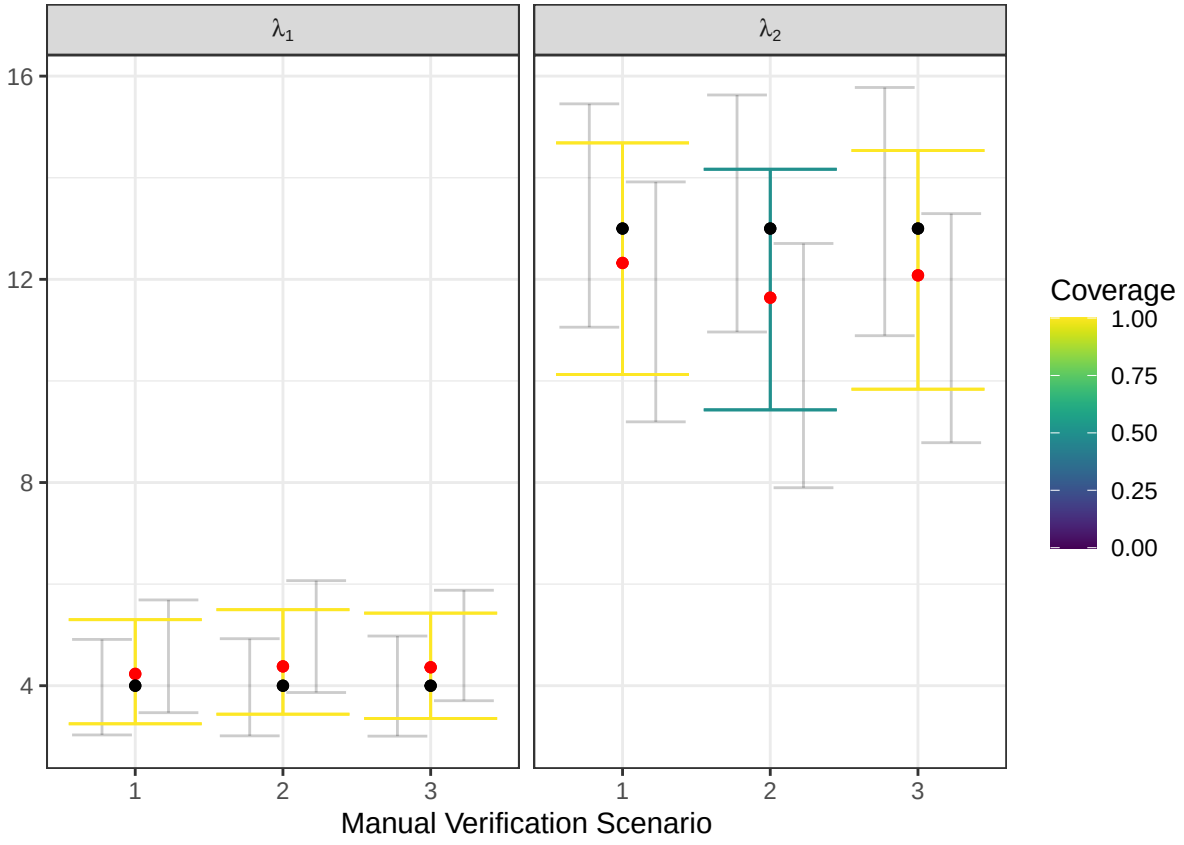
**Step 5: Visualize simulations**

Once the simulation study is complete, you can visualize the results using two functions, `visualize_parameter_group` and `visualize_single_parameter`. The first one 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 $\Theta$ classifier ID.
- `scenarios`: Which scenarios to visulaize?
- `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".

```
source("Summary Figures/visualize_sims.R")
visualize_parameter_group(sim_summary = sims_output,
                          pars = "lambda",
                          theta_scenario = 1,
                          scenarios = 1:3,
                          convergence_threshold = 1.6) # for illustration only!
```

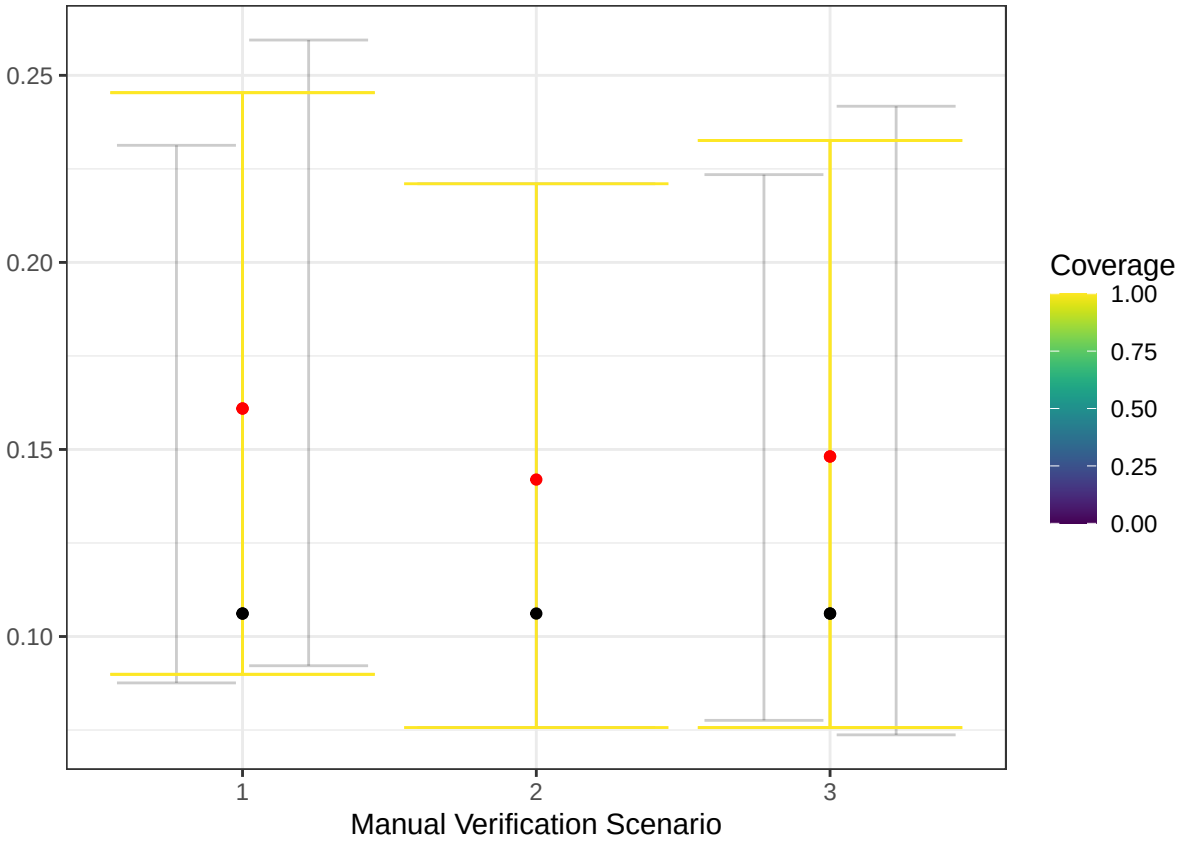


The features of the plot are as follows:

- 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:

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



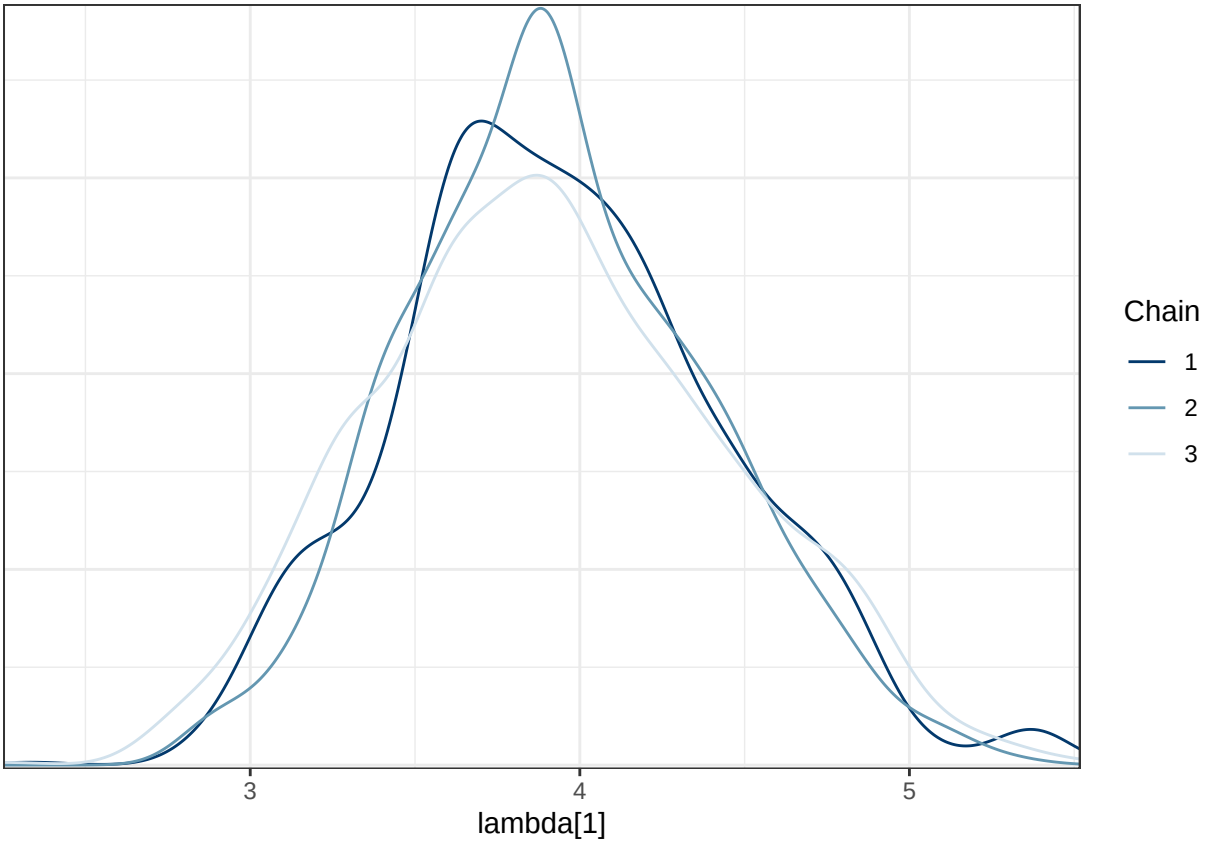

Note that in scenario 2, only one dataset yielded a fitted model where all parameters were below `convergence_threshold`. Because of this, the average posterior interval is the same as the individual grey error bar (which is hidden).

## Using saved datasets and model fits

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`:

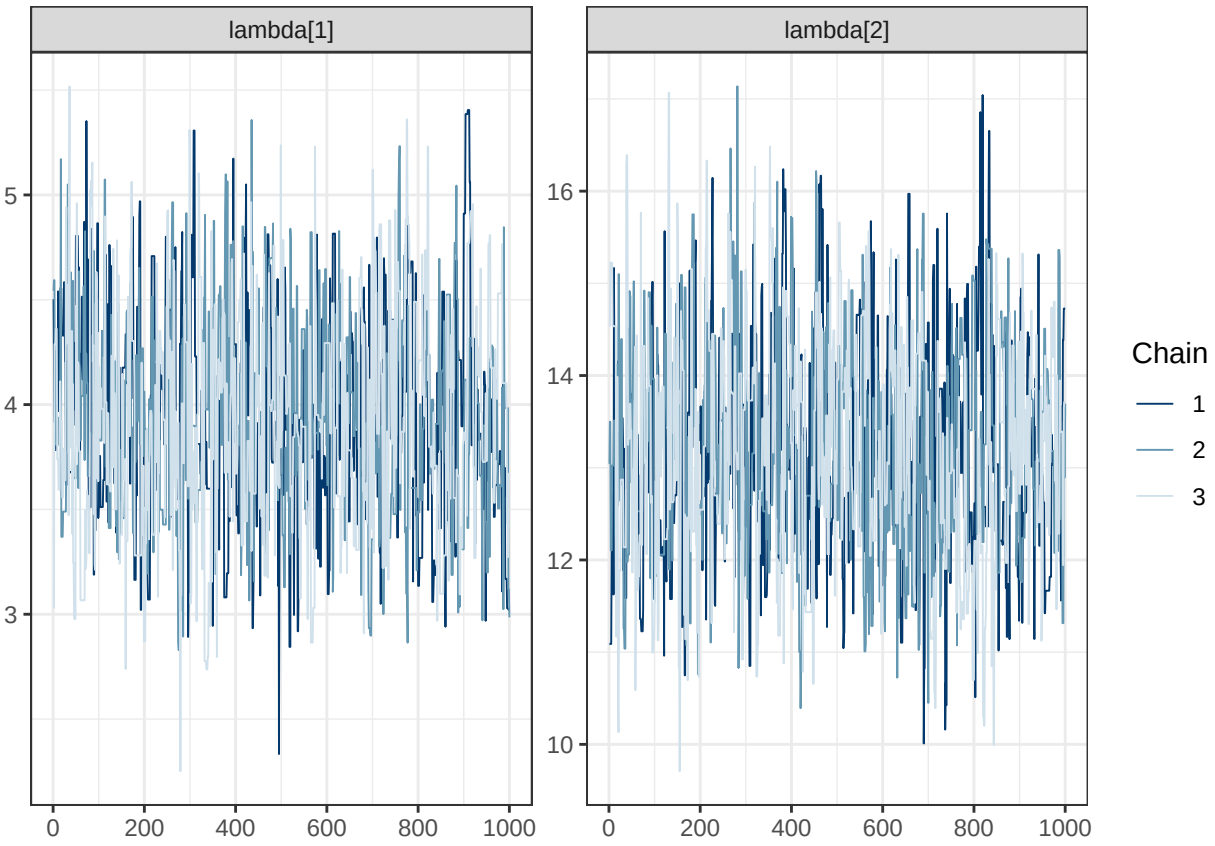
```
# read in fit object
fit_1_1 <- readRDS("Testing/Theta1/fits/fit_1_1.rds")

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




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

```
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/).