Vignette: Simulation studies with `ValidationExplorer` under a fixed-effort design type

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

2024-11-22

Abstract

Our vignette demonstrates the use of the `ValidationExplorer` package to conduct statistical simulation studies that explore the costs and inferential properties (e.g., near nominal coverage and/or minimal estimation error) of alternative validation designs. 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 aid informed management decisions using bioacoustic data in the presence of misclassification', which was submitted to the Applications series of *Methods in Ecology and Evolution*. In this vignette, we provide 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.

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

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

Contents

1 Introduction

Automated recording units (ARUs) provide one of the main data sources for many contemporary monitoring programs that aim to provide inference about status and trends for assemblages of species (Loeb et al. 2015). As described in "`ValidationExplorer`: Streamlined simulations to aid informed management decisions using bioacoustic data in the presence of misclassification" (hereafter, "the main text"), substantial practical interest lies in identifying cost-effective validation designs that will allow a monitoring program to obtain its measurable objectives. We believe that statistical simulation studies are a valuable tool for evaluating the relative merits of candidate validation designs prior to gathering and validating ARU data, and it is our goal for `ValidationExplorer` to provide those tools.

This vignette provides detailed demonstrations of the `ValidationExplorer` package under a fixed-effort design. 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 sample size. We refer to the random mechanism as the *design type* and the proportion as a the level of validation effort (LOVE). In our example, we assume a fixed effort (the design type) validation design, under which, x% of recordings obtained from the first visit to a site are validated by experts. The level of validation effort is controlled by the value of x. In the following section we consider five possible LOVEs and show an example of a fixed effort design.

2 Conducting a simulation study with a fixed effort design type

2.1 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 Section 3 of the main text:

- The measurable objective is to estimate relative activity parameters (denoted as λ_k) for each species with estimation error less than 1 call per night and with the expected width of 95% posterior intervals less than 3 calls per night.
- The monitoring program can pay their expert bat biologists to validate at most 4000 recordings. WE

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)	0.6331	5.9347
Lasiurus cinereus (LACI)	0.6122	4.1603
Lasionycteris noctivagans (LANO)	0.8490	14.2532
Myotis californicus (MYCA)	0.6972	6.1985
Myotis ciliolabrum (MYCI)	0.2365	11.8649
Myotis evotis (MYEV)	0.7036	2.4050

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).

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.

2.2 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)
```

Note: to knit this vignette, you may also need to install the `kableExtra` package.

2.3 Step 3: Simulate data {dataSimulation}

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:

```
# 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)
```

Additionally, `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)}))

print(Theta)
```

```
##              [,1]        [,2]       [,3]         [,4]         [,5]         [,6]
## [1,] 0.953670138 0.015886205 0.01346950 0.0005918829 0.0153509623 0.001031315
## [2,] 0.028614016 0.914794071 0.01941594 0.0205164961 0.0065938947 0.010065584
## [3,] 0.013669368 0.006878823 0.82604797 0.0969673863 0.0005128495 0.055923603
## [4,] 0.042623395 0.028935593 0.01156918 0.8327210780 0.0599628913 0.024187867
## [5,] 0.005590593 0.034119356 0.02036092 0.0639122608 0.8635266207 0.012490252
## [6,] 0.003122368 0.003831873 0.04087851 0.0742637611 0.0632647264 0.814638762
```

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)}))
print(another_Theta)
```

```
##            [,1]        [,2]        [,3]        [,4]       [,5]        [,6]
## [1,] 0.55731323 0.142858110 0.053100053 0.003946041 0.04273763 0.200044944
## [2,] 0.13437231 0.499939788 0.021421570 0.014098339 0.02959848 0.300569509
## [3,] 0.10172739 0.007228338 0.515224477 0.155895124 0.13383729 0.086087378
## [4,] 0.18412846 0.210913704 0.026614757 0.479425606 0.09220850 0.006708972
## [5,] 0.27654377 0.142140440 0.004483512 0.068078975 0.32570927 0.183044035
## [6,] 0.06086223 0.043479478 0.123594690 0.083448153 0.15034635 0.538269101
```

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:

```
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)
```

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

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

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

```
# 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 using the following code:

```
sim_data <- simulate_validatedData(
    n_datasets = 50,
    nsites = nsites,
    nvisits = nvisits,
    nspecies = nspecies,
    design_type = "FixedPercent",
    scenarios = c(0.05, 0.1, 0.15, 0.3, 0.5),
    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 by `scenarios = c(0.05, 0.1, 0.15, 0.3, 0.5)`. 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 the first visit to a site are validated, while in the second validation design 10% of recordings from the first 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:

- `full_datasets`: A list of length `n_datasets` with unmasked datasets (i.e., the datasets if all recordings were validated so that every recording has an autoID and a true species label). We opted to not save these datasets 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.
- `zeros`: A list of length `n_datasets` containing the true species-autoID combinations that were never observed at each site visit. For example, if, in dataset 10, no calls from species 1 were classified as species 3 on visit 4 to site 30, then the 10th entry of this list would contain a dataframe with a row corresponding to site = 30, visit = 4, true_spp = 1, id_spp = 3, with count = 0. These zeros are necessary for the model to identify occurrence probabilities and relative activity rates. 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.
- `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 output are given below. First, we examine the third simulated full dataset. Notice that in addition to the site, vist, 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 an true species-autoID pair. 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.

```
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
```

The next object output from `simulate_validatedData` is the list of `zeros`. These are the potential true species-autoID combinations that were not observed at each site-visit. Notice that `count=0` for all observations; for instance, we never observed species 1 being classified as species 2 on visit 1 to site 1, despite species 1 being present.

```
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
```

The final output object is the nested list of `masked_dfs`. 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 below, we are considering the 3rd dataset under validation scenario 4. Further, 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, 3, 5 and 6 were not selected for validation (because `true_spp = NA`), while recordings 2 and 4 were (both calls came from species 1).

```
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]])
```

```
## # A tibble: 6 x 11
## # Groups:   site [1]
##    site visit true_spp id_spp lambda   psi  theta     z count    Y. scenario
##   <int> <int>    <int>  <int>  <dbl> <dbl>  <dbl> <int> <int> <int>    <int>
## 1     1     1       NA      1   5.93 0.633 0.954      1     4    24        4
## 2     1     1        1      1   5.93 0.633 0.954      1     4    24        4
## 3     1     1       NA      1   5.93 0.633 0.954      1     4    24        4
## 4     1     1        1      1   5.93 0.633 0.954      1     4    24        4
## 5     1     1       NA      2   6.20 0.697 0.0289     1     1    24        4
## 6     1     1       NA      3  14.3  0.849 0.826      1    10    24        4
```

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 = 1:5)
```

```
## # A tibble: 5 x 3
##   theta_scenario scenario n_validated
##   <chr>          <chr>          <dbl>
## 1 1              1               54.8
## 2 1              2               93.9
## 3 1              3              135.
## 4 1              4              255.
## 5 1              5              410.
```

2.4 Step 4: 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 provides the user with information about the

warmup and 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.

As in the main text, we use a dataset with the lowest number of validated recordings, for which we expect the greatest number of iterations. In our example, this is scenario 1, in which an average of ≈ 55 recordings are validated per dataset. When we ran `tune_mcmc` with dataset 39 under scenario 1, we received a series of error messages, with the message that convergence was not reached in under 10,000 iterations. We have several options:

1. We could increase the number of iterations 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 (55 calls validated per dataset of size ≈ 3300) 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. We adopt this approach here and run `tune_mcmc` again under scenario 2, which has around 94 calls validated per dataset on average.

```
tune_list <- tune_mcmc(dataset = sim_data$masked_dfs[[2]][[25]], zeros = sim_data$zeros[[25]])
```

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

We can see from the output that minimum number of iterations is 2000 with a warmup of 1000:

```
tune_list$min_iter
```

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

```
tune_list$min_warmup
```

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

If we were to specify 10,000 iterations per dataset, each model fit would take approximately 2 minutes.

```
tune_list$max_iter_time
```

```
## Time difference of 2.180024 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.

To understand the MCMC in greater detail, we examine the `convergence_matrix` output:

```
tune_list$convergence_matrix
```

```
##                            warmup = 500 warmup = 1000 warmup = 2000 warmup = 5000
## post-warmup iters = 1000              1             1             1             1
## post-warmup iters = 2000              1             1             1             1
## post-warmup iters = 3000              1             1             1             1
## post-warmup iters = 4000              1             1             1             1
## post-warmup iters = 5000              1             1             1             1
```

This matrix shows which combinations of warmups and post-warmup iterations yielded a set of markov chains that exhibited evidence of convergence (as indicated by a value of 1 in the corresponding entry). For example, our fitted model shows approximate convergence after running the MCMC for 3000 iterations, and discarding the first 2000 as warmup (`warmup = 2000`, `post-warmup iters = 1000`). This gives an idea of how to tune the MCMC.

We emphasize that these results 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, this approach may save time in the long run by avoiding the need to re-run simulations. We show this approach in the following section, where we specify that there should be 3500 iterations with the first 2500 discarded as warmup.

2.5 Step 5: 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:

```
# reduce the size of the list since validation scenario 1 is too low:
# we want scenarios 2-5 for fitting.
# Note that now scenario 1 in the output will correspond to validation scenario 2,
# scenario 2 in the output will correspond to validation scenario 3, and so on.
masked_dfs_reduced <- masked_dfs[2:5]

sims_output <- run_sims(
         data_list = masked_dfs_reduced,
         zeros_list = sim_data$zeros,
         DGVs = list(lambda = lambda, psi = psi, theta = Theta),
```

```
        theta_scenario_id = "FE", # for "fixed effort"
        parallel = TRUE,
        niter = 3500,
        nburn = 2500,
        thin = 1,
        save_fits = TRUE,
        save_individual_summaries_list = FALSE,
        directory = here::here("Vignette", "Fixed_Effort")
)
```

```
## Beginning scenario 1.

## 2024-11-22 15:48:17.146181

##   |                                                                      |

## Beginning scenario 2.

## 2024-11-22 16:40:24.556529

##   |                                                                      |

## Beginning scenario 3.

## 2024-11-22 17:30:31.410638

##   |                                                                      |

## Beginning scenario 4.

## 2024-11-22 18:21:22.194823

##   |                                                                      |
```

2.6 Step 6: Visualize simulations

Once the simulation study is complete, you can visualize the results using several functions. The most detailed functions are `visualize_parameter_group` and `visualize_single_parameter`.

The plots output from these functions have the following features.

- 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

`visualize_parameter_group` 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.

```
# correct the scenario numbers in the output to match the validation scenario
# numbers
sims_output$scenario <- sims_output$scenario + 1
```

```
visualize_parameter_group(sim_summary = sims_output,
                          pars = "lambda",
                          theta_scenario = 1,
                          scenarios = 2:5,
                          convergence_threshold = 1.05)
```

The output indicates that under all validation scenarios we considered (2-5), the expected inference for relative activity rates of species 1-4 and 6 meets our measurable objectives: the posterior interval width is less than three for these species and there is minimal estimation error. However, note that under validation scenario 2, fewer of the models converged within 3000 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 for this species. We can see this more clearly by examining output through alternative visualization functions.

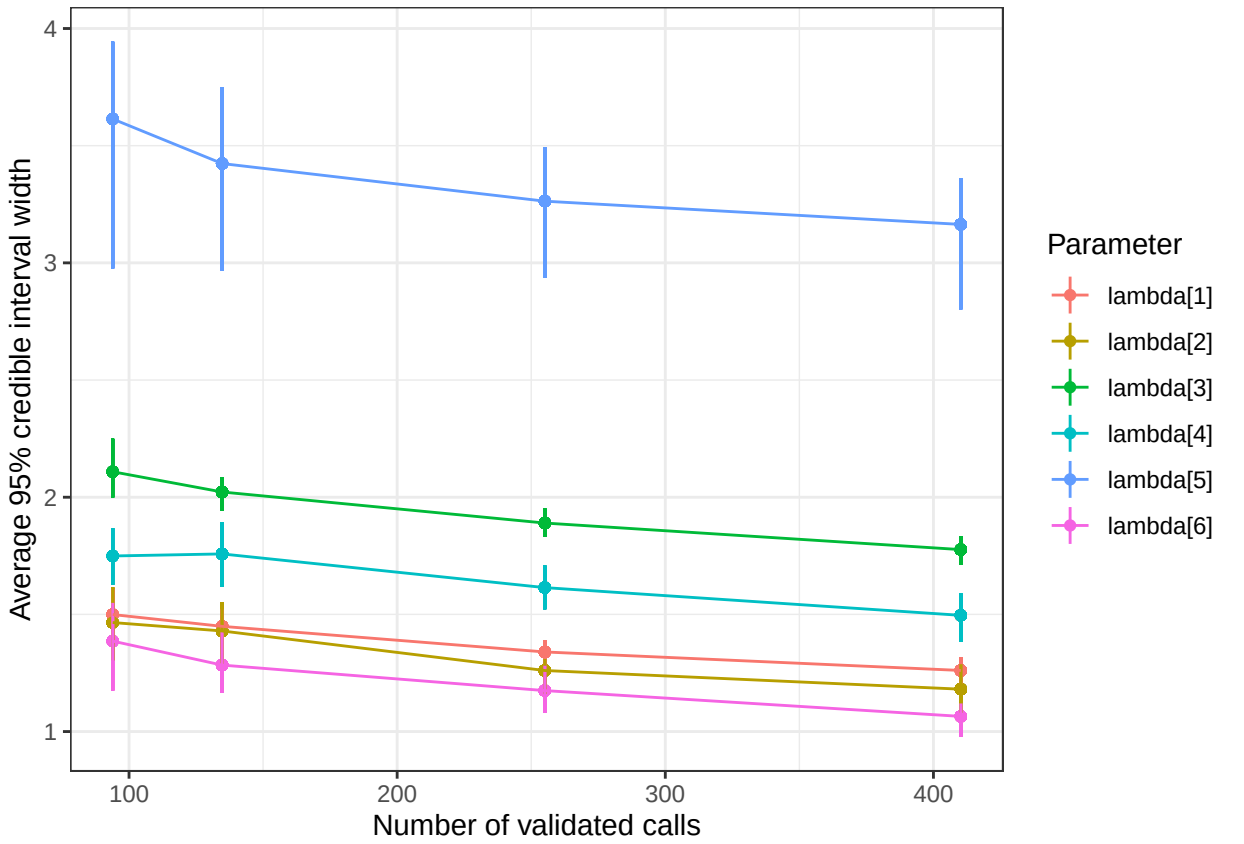
A first step would be to use `visualize_single_parameter`, which takes the same arguments as the previous visualization function:

```
visualize_single_parameter(sims_output, par = "lambda[5]",
                           theta_scenario = 1,
                           scenarios = 2:4,
                           convergence_threshold = 1.05)
```

The greater detail of `visualize_single_parameter` 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 larger than 3, it is not substantially larger in validation scenario 4. This can also be seen using the `plot_width_vs_calls` function, 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:

```
s <-  summarize_n_validated(
  data_list = sim_data$masked_dfs,
  theta_scenario = "FE",
  scenario_numbers = 2:5
)

plot_width_vs_calls(
  sim_summary = sims_output,
  calls_summary = s,
  regex_pars = "lambda",
  theta_scenario = "FE",
  scenarios = 2:5
)
```



We provide analogous functions `plot_coverage_vs_calls` and `plot_bias_vs_calls` taking identical argu-

ments 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.

3 Conclusion

We have demonstrated the use of the `ValidationExplorer` package. Note that with any simulation study results are conditional on the settings and assumptions. In the case of the count-detection model framework, the assumptions are

- The occurrence of species within a site are independent; the presence of one species carries no information about the presence or absence of another.
- For any one species, its occurrence at one location is independent of its occurrence at any other location (independence across sites).
- Visits to a site (i.e., detector nights) are independent.
- Recordings within the same site-visit are independent. This holds regardless of whether a recording is validated or remains ambiguous (only has an autoID label).
- The count of recordings generated by a species in a single detector night is a Poisson random variable.
- All species in the assemblage can co-occur and are capable of being confused (no structural zeros in the classification matrix).

The settings include the number of datasets used in the simulations, the assumed characteristics of the species assemblage (i.e., the values for each ψ_k and λ_k) and classifier (Θ), and the number of sites and visits. Our hope is that the ValidationExplorer provides a useful tool for assessing the merits of various validation designs, so that effort can be thoughtfully assigned based on program goals and monitoring objectives. Ultimately, we hope that this software streamlines the bat acoustic workflow and allows for efficient processing of information.

References

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.