BePhyNE Guide: Fitting your data to BePhyNE

Sean McHugh

2025-08-04

BePhyNE intro

Hello and thank you for checking out BePhyNE (fka PhyNE). This guide will show how to fit a set of spatial Presence and Absence/PsuedoAbsences with corresponding environmental data (ex WorldClim or envirem) to BePhyNE and recover estimates of unimodal symmatric response curves with 3 parameters: optimum, breadth, and (optionally) tolerance. For each environmental predictor for each species and estimates of the the evolutionary model parameters. Each curve evolves under a separate multivariate brownian motion (mvBM) where the optimum and breadth (and tolerance optionally) evolve given a an ancestral root mean and variance covariance matrix describing the rates of evolution and covariance between each response curve parameter

NOTE: all environmental predictor response curves are currently set as uncorrelated, if trying to estimate phylogenetic niche in temperature and precipitation, the impact of precipitation and temperature on occurrence would be unlinked (no correlation estimated in species specific ENMs) and also the covariance in response curve parameters are not modeled across predicotrs, only between (such that there is a correlation between precipitation optimum and breadth but not precipitation optimum and temperature optimum) Covariation in these predictors is very interesting and important next step in improving this models and other similar ones.

#Getting Start: Get the Package

lets install the package and

```
library(devtools)
devtools::install_github("sean-mchugh/BePhyNE")
library(BePhyNE)
```

Getting started: Data Format

To run BePhyNE you need a dataframe loaded in from a file format such as a ".csv" (can be loaded in from an excel spreadsheet saved as a.csv file usually) with a specific column order start including: - species name (this must be the same exact name as the corresponding tip label in your phylogeny), - P/A value (either a 1 for presence or 0 for absence), - Environmental predictors (one per row)

You can have columns before these (such as latitude and longitude) the user needs to specify the name of the species column, P/A value column, and predictor columns

This on the back end gets formatted as a list of lists, were each list includes the species name, the vector of P/A, and vectors for the corresponding environmental predictor values.

Lets in load in the example file which contains Presence absence data for 98 species of Plethodontid salamanders from Eastern North America (ENA_Pleth in a lot of the code) we have two macroclimate

predictors from worldclim: average annual precipitation (mm) and average annual temperature (degrees Celsisus)

most climatic variables you will want to scale before fitting with BePhyNE. When loading in your occurrence data you can either provide your own scaling object (in this case we do so, as we include scaling from a much larger set of climatic data), otherwise “format_BePhyNE_data” will scale the data if not other scaling is provided.

```
tree = ENA_Pleth_Tree
pa_data   = read.csv("~/Manuscripts/BePhyNE/Vignette_materials/Pleth_data_vignette.csv")
sp_col    = "species"
occ_col   = "PA"
env_preds = c("bio12", "bio1")
Npred = length(env_preds)
Ntips = length(tree$tip.label)

data_obj   = format_BePhyNE_data(pa_data, tree,sp_col, occ_col, env_preds, scale_atr=NA)
data_final = data_obj$data
sets_full  = suppressWarnings(separate.data(data_final,ratio = .5))
scale_atr  = data_obj$scale
```

So we have a dataset, now it is time to put it into a form BePhyNE can read, and then start setting up the details for our MCMC run. BePhyNE reads a list of lists of your PA dtaa and occurrences over each species such that for one species (the way ENA_Pleth_PA is converted to data_final makes column order essential) (NOTE: Lat and Lon are removed at this stage) data_final[[1]] $speciesname $ vector of y (1’s and 0’s for presences and absences) $ vector X1 $ vector X2 $ vector Xetx

Setting up the MCMC:Choose Your Priors

Now that we have our dataset we can start the exciting task of setting our priors, we will inform our parameters with informative/weakly informative priors (entirely uninformative uniform priors can lead to some weird behavior especially for niche breadth and mvBM pars). “make_all_priors is preloaded with presetted values (explictly written out in the arguements).

Since tolerance are only estimated from the glm, we will put priors on each tip. with the mean being a max likelihood estimate for tolerance Note what we are doing here is a little janky/circular as we are using max likelihood GLM(NOTE) estimates as the means, this is just a weak prior to place it somewhere between .05 and .99. You could use a flat prior but it will result in BePhyNE taking much longer to mix.

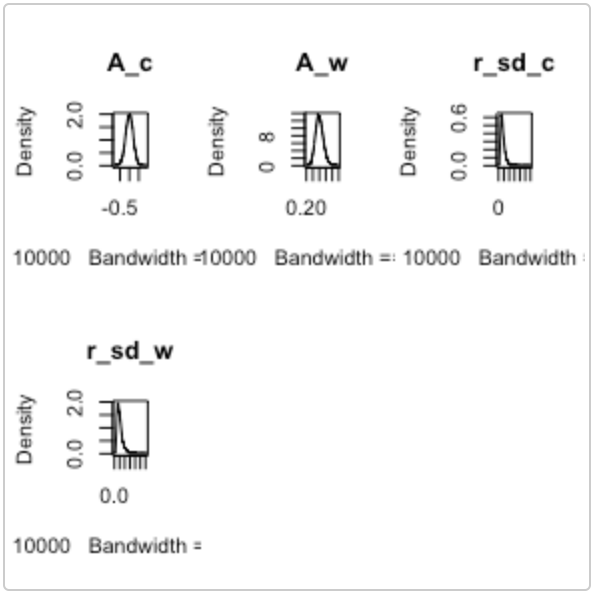
we do not believe this actually reflect prevalence, but instead error in sampling of the occurrences (broad scale gbif data) and generated absence data (using target group absences filtered to an equal number within the species IUCN expert range map and double the number of presences outside the expert range), see the pre print for details

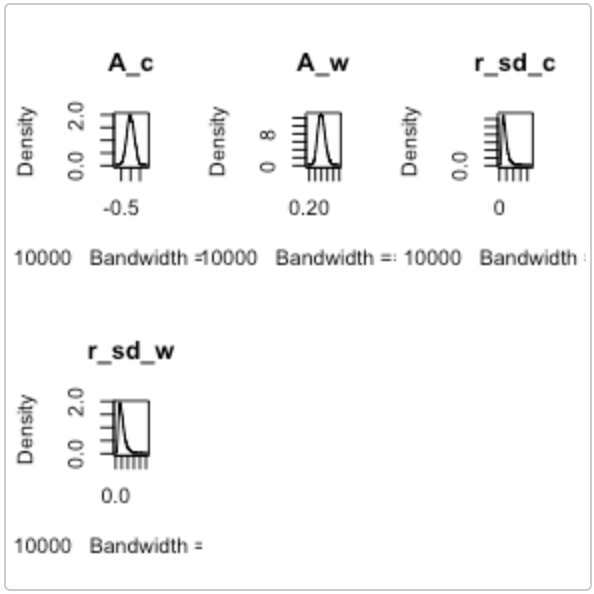
(NOTE) Generalized linear model: just the function we are fitting for each individual species response curve in PhyNE (logit(y) = B0 + B1 * X1 + B2 * X2)

```
 bd_range = c(0.1, 1.5)
bd = bd_range[[2]] - bd_range[[1]]

Prior_scale = make_all_priors(Npred, length(tree$tip.label),
```

```
                                bd_range = bd_range,
                                # ── μ (root) priors ──────────────────────────────────────
                                root_opt_mean         = rep(0,        Npred),
                                root_opt_sd           = rep(0.2,      Npred),
                                root_brdth_meanlog    = rep(log(.3),  Npred),
                                root_brdth_sdlog      = rep(0.1,      Npred),
                                # ── σ (log-normal) hyper-priors ──────────────────────────
                                sigsq_opt_meanlog     = rep(log(bd),  Npred),
                                sigsq_opt_sdlog       = rep(0.5,      Npred),
                                sigsq_brdth_meanlog   = rep(log(bd),  Npred),
                                sigsq_brdth_sdlog     = rep(0.5,      Npred),
                                # ── heights (default hard-coded) ─────────────────────────
                                heights_by_sp         = sample(.95, size = Ntips, replace =
        TRUE),
                                # ── constants forwarded to makePrior_ENE ─────────────────
                                r = Npred, p = 1,
                                plot = TRUE
)
```





Setting up the MCMC:Choose Your Starting Values

To get start values we simulate parameters of our model, including individual species response curves and mvBM pars, however because we only can set what the mvBM pars are and the tip responses are subsequently simulated, we dont have a lot of control over what the tip values will be and some potential starting values could be so far off they cause the likelihood function to return nonsensical values, breaking the MCMC.

```
startPars = get_starting_values(Prior_scale)[[1]]
```

Setting up the MCMC:Choose Your Tuning Parameters

Next you need to set your tuning parameters, how often do we want to be making proposals on any specific parameter, we set these through trial and error on our scaled predictors, each tip response par and mvBM par has its own tuning par in this setup, each response curve proposal type (center_lide, center_mult, width_slide, etc) has its own tuning par

```
move_details = make_tuning(tree = tree, pred = Npred)
```

Now you have all of the pieces to run the MCMC, you have an option to either write the MCMC to a file or store the chain as a giant list of lists. We recommend writing to a file and subsequent post porocessing code will be for outputs from the written log files.

An added benefit to writing the MCMc to file is the outputted log files can be read by user-friendly software such as Tracer.

```
#

sparse_sp=F
iterations=1000
trim_freq= iterations/1000
burnin=0
chain_end=(iterations-(burnin))/trim_freq
filename="~/Downloads/BePhyNE"

results = BePhyNE_MCMC(tree
                       ,pa_data = sets_full$training
                       ,Prior_scale
                       ,startPars
                       ,move_details
                       ,iterations = iterations
                       ,trim_freq  = trim_freq
                       ,write2file = T
                       ,filename
)
#> iterations: 190
#> acceptances: 100
#> acceptance ratio: Inf
#> move utilized: Height_Multiplier
#> iterations: 376
#> acceptances: 200
#> acceptance ratio: 19.74641
#> move utilized: R_corr
#> iterations: 589
#> acceptances: 300
#> acceptance ratio: 3.722386e+253
#> move utilized: Width_Multiplier
#> iterations: 779
#> acceptances: 400
#> acceptance ratio: 2.216436e+97
#> move utilized: Height_Slide
#> iterations: 975
```

```
#> acceptances: 500
#> acceptance ratio: 8.482769e+133
#> move utilized: Width_Slide
#> iterations: 1001
#> acceptances: 512
#> acceptance ratio (not accepted): 2.116884e-07
#> move utilized: Center_Slide
```

To summarize the MCMC output we have provided a small function to read the output into a dataframe, and another to summarize the dataframe into a set of effective sample sizes (ESS), HPD intervals, and posterior medians. The ESS and HPD have the same labelling as the log file which currently is still a little messy.

PArameters are organized by respective predictor index and traits are labeled by index (1= optimu, 2= breadth, 3=tolerance) (ex: pred_1_A1 is the root mean optimum for the first predictor, in this case bio12).

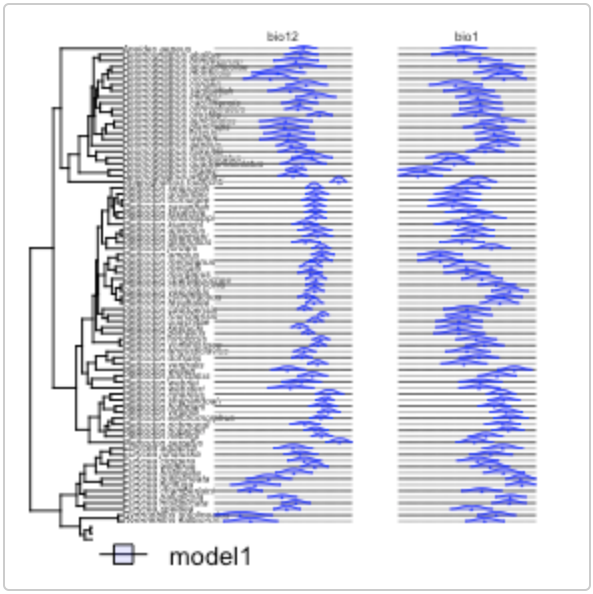
Traits are messy too, the are lablled as a compact integer, the first digit being the trait and the subsequent ones indicating species. (ex: pred_1_dat.trait11 is the optimum for bio12 in species one in terms of the tree$tip.label ordering). Future updates will be focused on cleaning this up, and plots currently swap these indices out for the correct names.

```
logdf = read_BePhyNE_log(file_name = paste0(filename, ".pars.log"))

log_summary = summarize_logdf(logdf)
```

Now let plot what the posterior median response curves look like using ridge plot.

```
plot_summary_ridgeplot (tree,
                        log_summary  ,
                        model_names       = paste0("model", 1),
                        predictor_names   = env_preds,   # character vector of predictor
        names
                        scale_atr         = scale_atr,         # list with $center and $scale
        (same length as predictors)
                        curve_colors      = rep(make.transparent("blue", 255/255),1),
                        curve_fill_colors = rep(make.transparent("blue", 30/255), 1),
                        line_types        = rep(1, 1),
                        xlims = NULL
)
```



We also now have ancestral state estimation included using Bruce Stagg Martin's excellent contsimmap package (much thanks to Bruce for helping get this working!). Currently for speed we set the number of sims to 10, but you will likely want many more (closer to 100-1k). When plotting, user you scale_atr from before to transform values back to climate space.

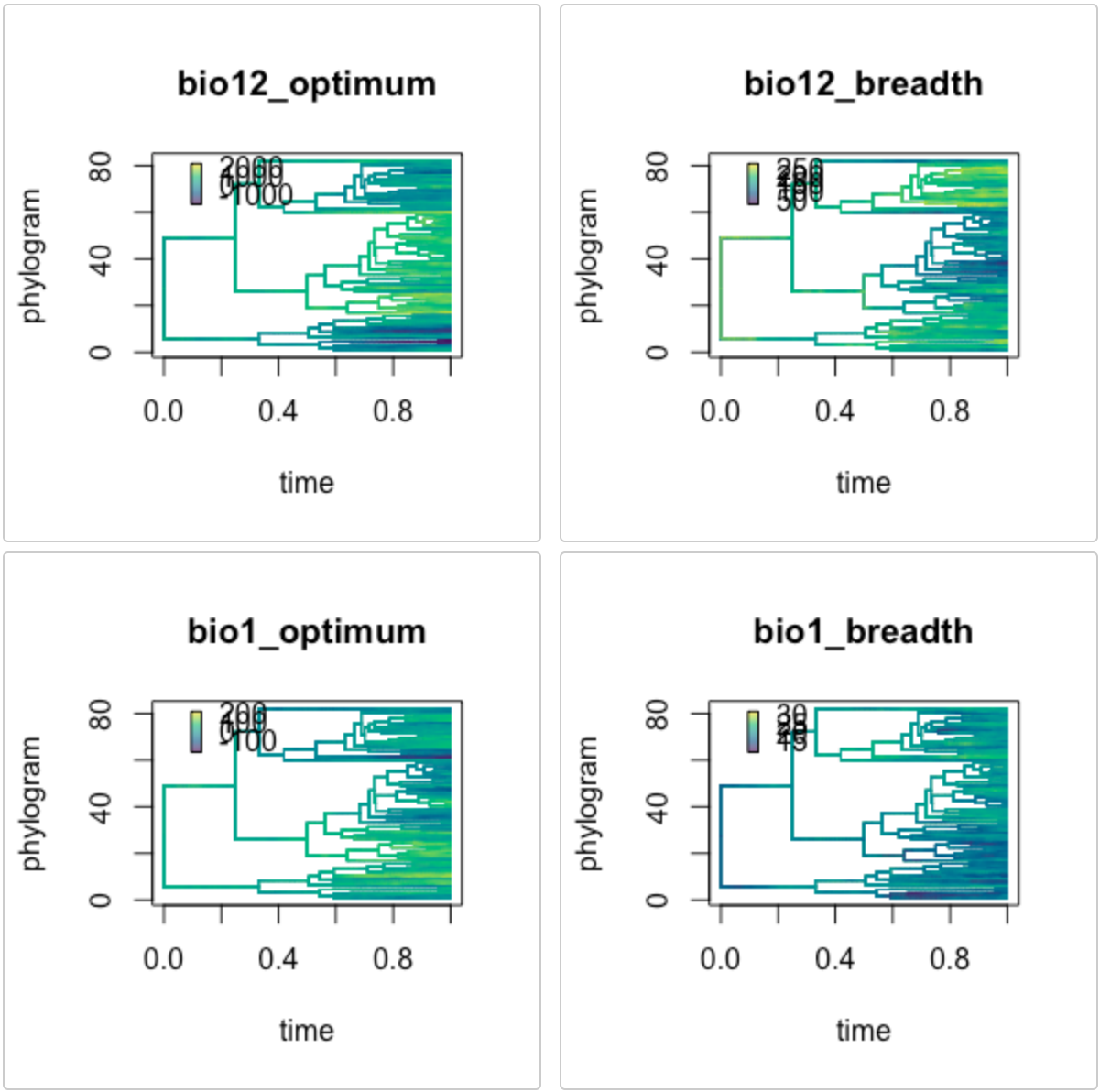
```
#####stochastic character mapping################
#not working great yet, can only plot characters in BM space, and cant o verwrite character
        values with nichespace characters
#devtools::install_github("https://github.com/bstaggmartin/evorates")
#devtools::install_github("https://github.com/bstaggmartin/contsimmap")

library(evorates)
library(contsimmap)
#>
#> Attaching package: 'contsimmap'
#> The following objects are masked from 'package:evorates':
#>
#>     alter.cols, anc.edges, des.edges, edge.ranges, root.edges,
#>     sis.edges, tip.edges
#> The following object is masked from 'package:foreach':
#>
#>     accumulate

char_names = c("bio12_optimum","bio12_breadth",
               "bio1_optimum","bio1_breadth")
contsimmaps = make_simmaps_BePhyNE( tree,
                                    logdf,
                                    nsims = 10
                                    ,char_names= char_names
                                     )
#> [1] 761
#> [1] 761
#> [1] 779
#> [1] 779
```

```
#> [1] 5
#> [1] 5
#> [1] 954
#> [1] 954
#> [1] 247
#> [1] 247
#> [1] 459
#> [1] 459
#> [1] 447
#> [1] 447
#> [1] 28
#> [1] 28
#> [1] 883
#> [1] 883
#> [1] 325
#> [1] 325
```

```
plot_BePhyNE_simmap(contsimmaps, scale_atr = scale_atr)
```



Next you can do some model adequacy testing. predict_stats_list will generate a set of Receiver Operator Characteristic statistics, from which we can extract and plot area under the curve (AUC) stats per species.

```
#####AUC#################

predict_stats_list= AUC_posterior_median(log_summary, sets_full$predicting)

plot_AUC_treebarplot(tree, predict_stats_list )
```