User manual for HMSC

Anna Norberg[1], Gleb Tikhonov[1], F. Guillaume Blanchet[2], Nerea Abrego[3], and Otso Ovaskainen[1]

[1]Department of Bioscience, University of Helsinki
[2]Départment de biologie, Université de Sherbrooke
[3]Centre for Biodiversity Dynamics, Department of Biology, Norwegian University of Science and Technology

April 25, 2017

Contents

1 General information

The Hierarchical Modelling of Species Communities (HMSC) framework described in [1] has been implemented as MatLab- and R-packages. This manual aims to be a bridge between the help files of the packages (where more technical details are given) and the article (where the ecological interpretation of the results is discussed and the mathematical description of HMSC is given). For this, we provide step-by-step guidelines for replicating the results of the four case studies illustrated in the main article, as well as simulated examples. This particular version of the manual focuses exclusively on explaining the details of how the R package works.

This manual starts by giving the instructions for installing the packages, then describes the model structures and parameters defined within the HMSC framework and finalizes by instructing the user on how to perform the analyses in practice.

As we expect that the ultimate objective of the user is to apply the HMSC framework to her/his own data, we recommend the user to start by reading [1] where the capacity of the framework is discussed, then replicate the simulated examples which illustrate the use of the framework in a simplified setting, and eventually replicate the real data examples, which also illustrate somewhat more complicated study designs or types of analyses.

2 Getting started

2.1 Installing the software

The `HMSC` R package is available in its source (.tar.gz) and compiled versions (for Mac OS X, .tgz and Windows .zip). Many parts of the R package have been implemented in C++11 and for this reason it is simpler to use the compiled version of the R package. If the user wants to compile the R package, it is necessary also to have installed the R packages `coda`, `Rcpp` and `RcppArmadillo`. After it is downloaded, the R package can be installed using the `install.packages` function.

2.2 A suggestion for the general workflow

We recommend the user to conduct HMSC analyses by writing a script (or set of consequent scripts), that consists of all the commands needed to run the analyses, from importing the data to the production of the result figures and tables. In this way, the user will more easily ensure the reproducibility of the results. As illustrated in the example scripts below, a typical script includes two parts: one which relates to model fitting and another one which relates to the post-processing of the results.

The first part of the workflow consists of fitting an HMSC model. This includes

1. Setting up an HMSC object by defining the model structure (e.g. whether traits or random effects are to be included) and organizing the data (typically imported from files with standard R commands).

2. Defining the priors required for Bayesian inference.

3. Initiating the model parameters

4. Running the actual estimation scheme. This involves setting the Markov chain Monte Carlo (MCMC) sampler (e.g. by defining the number of iterations and how they will be thinned). As running the estimation scheme may take some time, we recommend the user to save the HMSC object to a file outside of R (e.g. called "model.RData"). This object includes the model structure, the data, and the full posterior distribution.

The second part of the workflow consists of reading the HMSC object from the saved file and post-processing the results. Unlike the model fitting procedure described above, this part of the workflow does not necessarily follow any specific order or include any compulsory components, as it is up to the user about the kind of output needed. But typically the user may wish to assess the convergence of the MCMC chains, produce posterior summaries (e.g. posterior means and quantiles) such as tables or plots, measure the explanatory power of the model, perform a variance partitioning among the fixed and random effects, and make predictions.

2.3 An overview of the structure of the rest of this manual

We first describe the general structure of the HMSC model as well as give an overview of the typical outputs that the user may wish to generate. We then go through two simulated and four real case studies, that we hope will help to serve as illustrative examples that will help the user in analysing his or her own data.

3 The general structure of the HMSC model

Figure 1 gives a graphical overview of the statistical model, the mathematical description of which is described by [1].

It is important to note that not all of the data types illustrated in Figure 1 are necessary to build an HMSC model. Technically, the only compulsory component for fitting the HMSC models is the occurrence matrix for at least one species and an intercept. In this case, the model will relate to estimating the mean occurrence (abundance or prevalence) of the species. However, for ecologically meaningful analyses, the minimal set of data needed are either of the following:

1. The occurrence matrix for one species as well as the environmental covariate matrix, in which case the HMSC model corresponds to a traditional single-species model.

2. The occurrence matrix for several species, in which case the HMSC model corresponds to a model-based ordination.

Although such analyses are possible with the `HMSC` R package, it is not primarily meant for these special situations, for which many other kinds of software are available. As the availability of the data types described in Figure 1 increase, deeper analyses and consequently deeper ecological insights become possible — and this is what the `HMSC` R package is really meant for.

3.1 Input data

We index the sampling units by $i = 1, \ldots, n_y$; the species by $j = 1, \ldots, n_s$; the environmental covariates by $k = 1, \ldots, n_c$; the traits by $t = 1, \ldots, n_t$; and the random effect levels (e.g. sampling units, plots, years, ...) by $r = 1, \ldots, n_r$ to be estimated.

The main input data (Figure 1) are contained in the following four matrices:

$\mathbf{Y}$ an $n_y \times n_s$ matrix of the response data (called `Y` in the `HMSC` R package)

$\mathbf{X}$ an $n_y \times n_c$ matrix of the environmental covariates (called `X` in the `HMSC` R package)

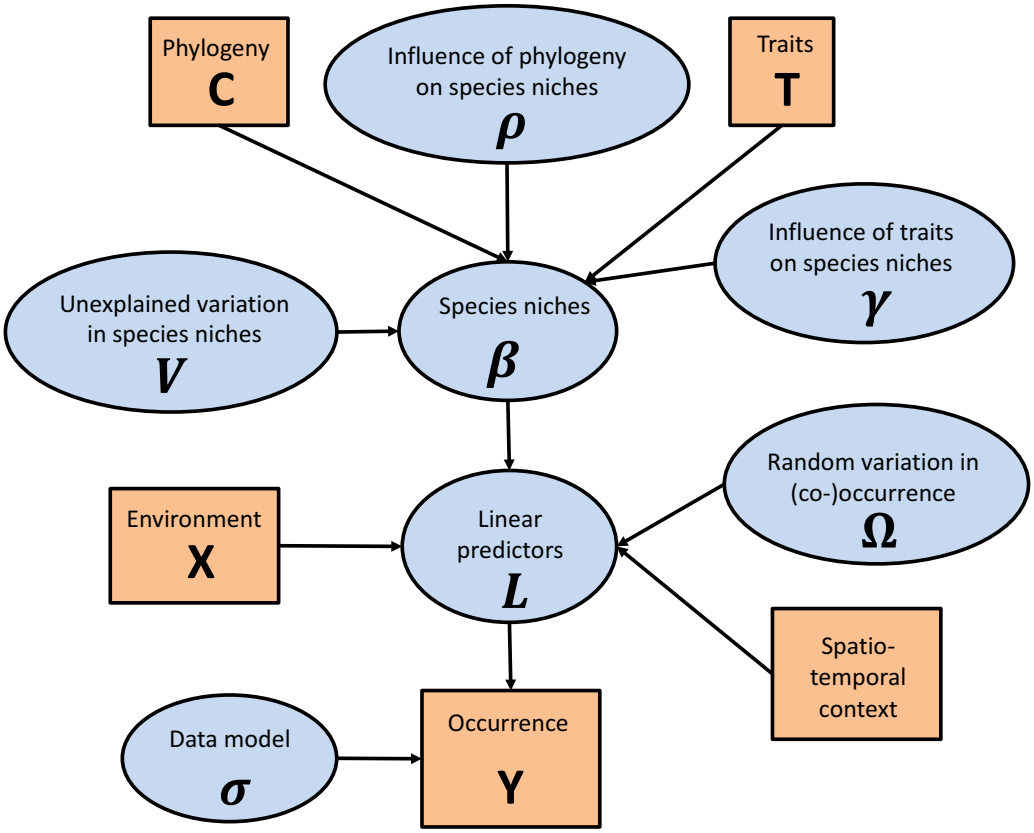

Figure 1: A graphical summary of the statistical framework. The orange boxes refer to data, the blue ellipses to parameters to be estimated, and the arrows to functional relationships described with the help of statistical distributions. See [1] for more details.

$\mathbf{T}$ an $n_s \times n_t$ matrix of the traits (in the `HMSC` R package, `Tr` is the transpose of $\mathbf{T}$)

$\mathbf{C}$ an $n_s \times n_s$ matrix of the phylogenetic correlations $c_{jj'}$ (called `Phylo` in the `HMSC` R package)

Additionally, the HMSC-framework accounts for random variation in species occurrence and co-occurrence, for which the spatio-temporal context of the study works as input data. The description of the structure of the study design is give in the following matrices:

$\mathbf{\Pi}$ an $n_y \times n_r$ matrix of the units π_{ir} to which the sampling units (rows of the $\mathbf{Y}$ matrix) belong to; for example, if r corresponds to a plot-level effect, π_{ir} is the plot to which the sampling unit i belongs to. The number of units at each level is denoted by $n_{p(r)}$ (called `Random` in the `HMSC` R package)

$\mathbf{xy}(r)$ a $n_{p(r)} \times d$ matrix containing the spatial (or temporal) coordinates of the units at level r. This matrix is defined for those levels r for which the study design has an explicit spatial (or temporal) structure. These coordinates can be of any dimension $d = 1, 2, \ldots$, typically with $d = 2$ for spatial coordinates and $d = 1$ for time. These coordinates are translated into a distance matrix using Euclidian distance (called `Auto` in the `HMSC` R package).

The user also needs to define the statistical distribution to use to construct the model. In the current version of the framework, there are four link functions/error distributions available: 1) `gaussian` for normally distributed data, 2) `probit` for binary data, and 3) `poisson` and 4) `overdispersedPoisson` for count data. To explain these mathematically, we denote by L_{ij} the linear predictor for sampling unit i and species j.

1. The `gaussian` model is defined as $y_{ij} = L_{ij} + \varepsilon_{ij}$, where $\varepsilon_{ij}\ N(0, \sigma_j^2)$
2. The `probit` model is defined as $\Pr(y_{ij} = 1) = \Phi(L_{ij})$, where Φ is the probit link function, i.e. the cumulative density function of the standard normal distribution $N(0, 1)$. The estimation procedure utilizes the mathematically equivalent formulation of $y_{ij} = 1$ for $z_{ij} \geq 0$ and $y_{ij} = 1$ for $z_{ij} < 0$, where $z_{ij} = L_{ij} + \varepsilon_{ij}$, and $\varepsilon_{ij}\ N(0, \sigma_j^2)$, where σ^2 is fixed to 1.
3. The `poisson` model is defined as $y_{ij} = \text{Poisson}(\exp(L_{ij} + \varepsilon_{ij}))$, where $\varepsilon_{ij}\ N(0, \sigma_j^2)$ and σ_j^2 is fixed to 0.01. The reason for including a small residual variance for the `Poisson` model (rather than setting it to zero, which would correspond exactly to the usual Poisson model) relates to the posterior sampling schemes we have used and is thus of technical rather than biological nature
4. The `overPoisson` models is defined as $y_{ij} = \text{Poisson}(\exp(L_{ij} + \varepsilon_{ij}))$, where $\varepsilon_{ij}\ N(0, \sigma_j^2)$, where σ_j^2 is estimated, unlike for the `poisson` model

3.2 Parameters to be estimated

For now, the posterior distributions of the parameters are compiled in `model`*results*`estimation`.

$\boldsymbol{\gamma}$ a $n_t \times n_c$ matrix describing the effects of the traits on the responses to covariates. It is referred to as `paramTr` and it is the transpose of $\boldsymbol{\gamma}$

$\mathbf{V}$ a $n_c \times n_c$ matrix of variation (in responses to covariates) among species which is not explained by traits. It is referred to as `varX`

$\boldsymbol{\beta}$ a $n_s \times n_c$ matrix describing responses of the species to the covariates. It is referred to as `paramX`

ρ (scalar) a parameter measuring the strength and sign of the phylogenetic signal on the species' responses to covariates. It is referred to as `paramPhylo`.

$\mathbf{\Sigma}$ a $n_s \times n_s$ diagonal matrix of residual variances (see the data models above), with diagonal elements σ^2. It is referred to as `varNormal` for Gaussian models

$\boldsymbol{\Omega}$ a $n_s \times n_s$ diagonal matrix of residual variances (see the data models above), with diagonal elements σ^2. It is referred to as `varNormal` species-to-species variance-covariance matrix. This association matrix is estimated separately for all levels listed in $\boldsymbol{\Pi}$ (e.g. one for sampling unit level and another one for plot level). The off-diagonal elements of these matrices model species-to-species associations. The diagonal elements model random variation within species, needed to account for the statistical dependency on cases where the data involves repeated samples from the same units. For example, if analysing the fungal data (with study design consisting of sampling units within plots within forests) for a single-species only, $\boldsymbol{\Pi}$ should involve the levels of plots and forests to make the analyses statistically valid, in the same way as one should include plot and forest as random effects in a usual GLMM analysis. If analysing the fungal data for multiple species, $\boldsymbol{\Pi}$ may or may not involve also the sampling unit level, depending on whether the user wishes to estimate the species-to-species association patterns at this level. Technically, these association matrices are estimated using a latent variable approach. The association matrix $\boldsymbol{\Omega}$ at level r is defined as $\boldsymbol{\Omega} = \boldsymbol{\lambda}\boldsymbol{\lambda}^T$ where $\boldsymbol{\lambda}$ is the $n_s \times n_{f(r)}$ matrix of the factor loadings. In the R package, this is referred to as `paramLatent` with $n_{f(r)}$ being the number of latent factors (which is also estimated). The latent factors are included in the $n_{p(r)} \times n_{f(r)}$ matrix $\boldsymbol{\eta}$. In the R package, this is referred to as `latent`. For spatial (or temporal) latent factors, we assumed a exponentially decreasing correlation structure, with spatial scale of factor h at level r denoted by α_{rh}. In the R package, this is referred to as `paramAuto`

4 Typical model outputs

After a HMSC model has been fitted and thus the posterior distribution of the parameters estimated, the user can output the results in various forms. Most importantly, the user has access to the full posterior distribution, and can thus post-process the results in the way she/he likes. To aid a user with limited experience in programming or Bayesian inference, `HMSC` also provide a link with object of class `mcmc` from the `coda` package making it possible to evaluate the convergence of MCMC sampling scheme. `HMSC` also provides different kinds of summaries for predicting and interpreting the results. The exact shape and amount of output that can be generated will vary depending on the components that the fitted model included. In this section, we introduce some standard output types that the reader may wish to generate. In the context of the real data examples, we illustrate some further kinds of outputs that are specific to particular study designs.

4.1 MCMC diagnostics

With MCMC-based Bayesian inference, it is necessary to examine the mixing of the chains, i.e. whether the MCMC chains have become stationary and are long enough to yield a representative sample from the posterior. One simple way is to assess the convergence of the parameters by a visual inspection of the MCMC trace plots. For example, Figure 2 shows a trace plot of the parameters. In this trace plot, the chains appear to be converging properly. First of all, there is no obvious transient phase, as the parameter values in the beginning of the chains cover a similar range as in the end of the chains. Second, while the chains may show some level of autocorrelation (consecutive values are more similar to each other than values far away), the chains move many times up and down, and thus contain several independent samples from the posterior. As it is not desirable to include auto-correlated samples (they do not provide additional information and increase e.g. file sizes), it is usual to thin the chains after the sampling (as done in Figure 2). To obtain a highly accurate sample of the posterior, the chain could be ran e.g. 10 times longer, so up to 100,000 iterations, and then thinned, e.g. by a factor of 100 to produce 1000 essentially independent samples from the posterior.

It may also be worth it to study the density distribution of the MCMC runs for each parameters to make sure the parameters are properly estimated. Figure 3 presents an example of these plots. In theory, the distribution of each parameter is expected to follow a bell-shape density curve. Visually, a good MCMC run should be stable with regard to the trace plot (Figure 2) and should have a bell-shape

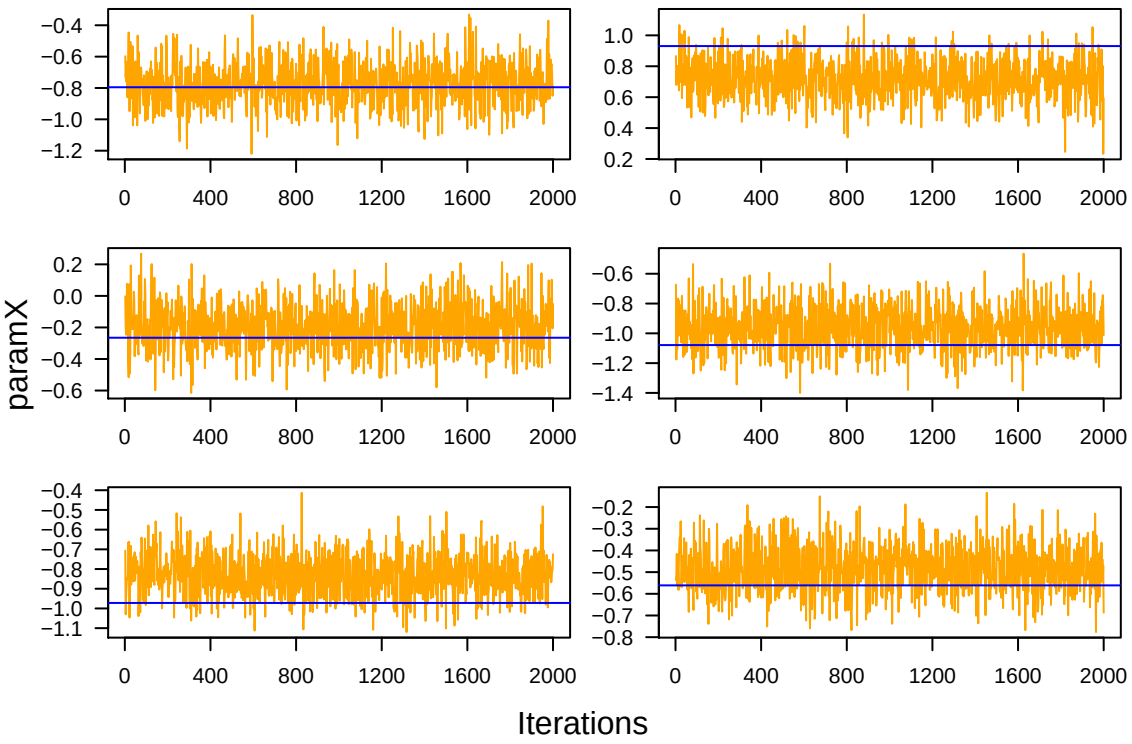

Figure 2: MCMC trace plot (orange line) illustrating the mixing chains of a set of parameters. The x-axis shows the MCMC iteration, and the y-axis the parameter values at each iteration, each panel is associated to a specific parameter. The blue lines correspond to the true parameter values (usually not known, but as this example is based on simulated data, they are known). The MCMC chain was ran for a total of 5000 iterations, of which the first 3000 were burning iterations and then thinned to include only every second sample, thus for each estimated parameter 1000 distinct values are plotted. This plot was constructed using `traceplot` of the `coda` package.

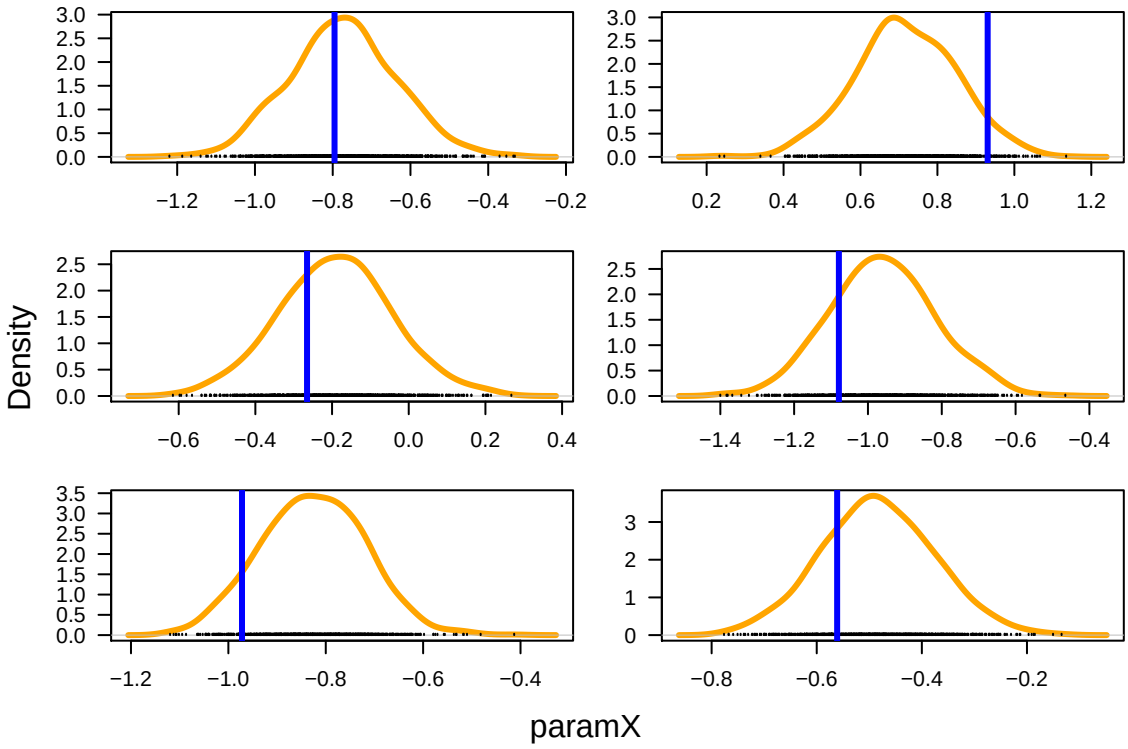

Figure 3: MCMC density plot (orange line) illustrating the density distribution of the mixing chains of a set of parameters. The x-axis shows the distribution of each paramters and the y-axis the density for parameter values, each panel is associated to a specific parameter. The black dots represent the MCMC iterations. The blue lines correspond to the true parameter values (usually not known, but as this example is based on simulated data, they are known). The MCMC chain was ran for a total of 5000 iterations, of which the first 3000 were burning iterations and then thinned to include only every second sample, thus for each estimated parameter 1000 distinct values are plotted. This plot was constructed using `densplot` of the `coda` package.

density curve. The `densplot` function from the `coda` package was used to draw the density plots in Figure 3. Note that because trace plots and density plots are important and complementary tools to assess the quality of an MCMC run, when the `plot` function is used on an object of class `mcmc`, both the trace and the density plots are drawn.

Another way to check for convergence is to use diagnostic tests such as Geweke's convergence diagnostic (`geweke.diag` function in `coda`) and the Gelman and Rubin's convergence diagnostic (`gelman.diag` function in `coda`).

4.2 Parameter summaries

Instead of reporting the full joint posterior distribution (which includes information e.g. on posterior correlations among the parameters), one often wishes to report summaries for individual parameter, i.e. properties of so called **marginal distributions**. As we are working with Bayesian inference, instead of p-values and confidence intervals (that are typically reported in maximum likelihood estimation) one may compute e.g. posterior probabilities or credible intervals. In the Bayesian analysis, the posterior mean (or median) can be considered as the best point estimate. To assess the level of statistical support for whether probability of occurrence (or abundance) of a species increases or decreases with increasing value of a given environmental covariate, one can construct e.g. the 95% central credible interval of any particular parameter by computing the 0.025 and 0.975 quantiles of the MCMC run for this parameter. If this credible interval contains e.g. only positive values, one may conclude that the parameter is positive (e.g. occurrence of species increases with the covariate) with a 95% level of statistical support.

Although it is possible (and sometimes interesting!) to study the parameters in table format, a way to summarize marginal posterior distributions is to visualize them with, e.g., box plots (by using the `boxplot` function [Figure 4]) or violin plots (by using the `beanplot` function in the `beanplot` package [Figure 5]).

4.3 Association networks

A key aspect of the HMSC framework is that it enables the estimation of species-to-species association matrices $\mathbf{\Omega}$. These matrices are technically variance-covariance matrices, and they can be turned into correlation matrices $\mathbf{R}$ by the conversion $R_{j_1 j_2} = \Omega_{j_1 j_2} / \sqrt{\Omega_{j_1 j_1} \Omega_{j_2 j_2}}$. Correlations have the advantage that their values always range from -1 to 1, and thus they are easier to interpret than covariance values, for which the absolute values depend on the variances.

The variance-covariance and correlation matrices can be interpreted as species-to-species association networks, and they can be estimated at different levels in the case of a hierarchical study designs. Summaries for both the covariance matrices and correlation matrices (e.g. posterior means and quantiles) can be outputted as two kinds of plots, illustrated in Figure 6. In the association plots, red colour corresponds to positive species-to-species associations (two species co-occur more often than expected by random) and blue corresponds to negative associations (two species co-occur less often than expected by random). The user can set the threshold level of statistical support for the associations to be included, and then only species pairs with statistical support above the threshold level are shown. The association networks can be plotted as equivalent 'matrix plots' or 'circle plots' (also called chord diagrams). In Figure 6, we can see that after accounting for the environmental covariates, e.g. the species pair E-F is found more often together than expected by chance, whereas the species pair A-F is found less often together than expected by chance. As discussed more thoroughly in [1], these associations can be considered as hypotheses for positive (species pair E-F) and negative (species pair A-F) species interactions.

4.4 Explanatory/predictive power

In [1], we illustrated the explanatory and predictive power of the model with coefficient of determinations (R^2). Such values can be computed either for the same data for which the model has been fitted (training data) or for predictions related to independent data (validation data). These values characterize the match between predicted and true occurrences for the training/validation data. For normally distributed

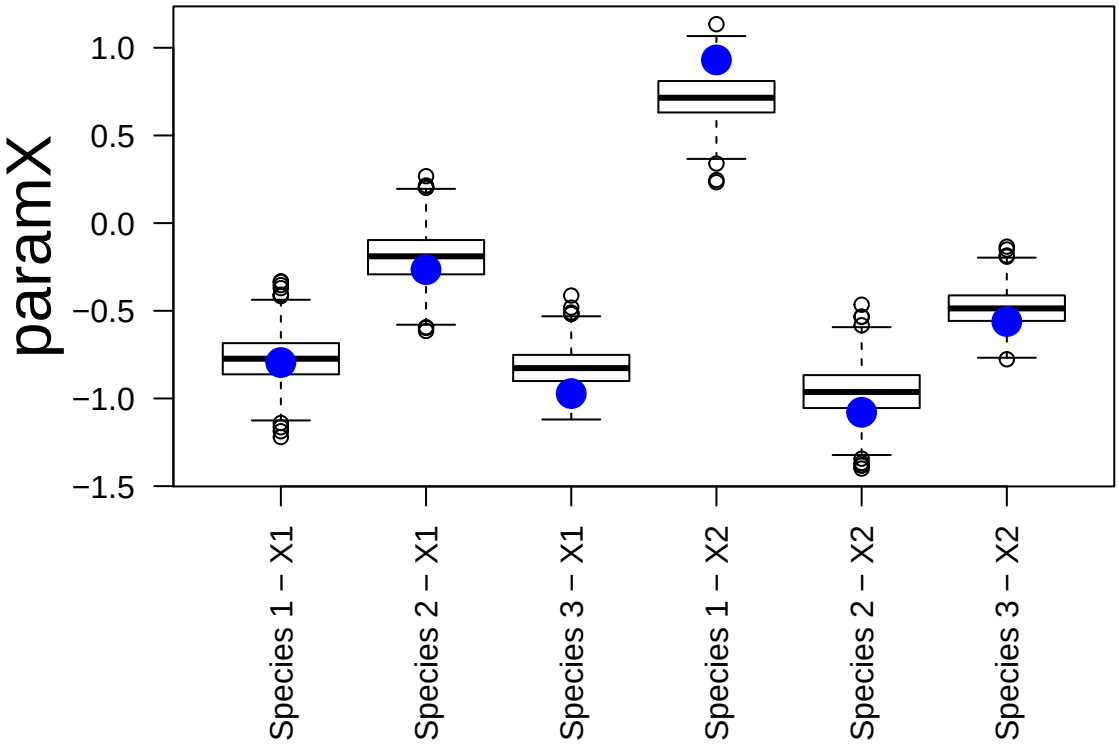

Figure 4: Box plot showing the marginal distribution of the `paramX` parameters. The blue points correspond to the true parameter values (usually not known, but as this example is based on simulated data, they are known). The MCMC chain was ran for a total of 5000 iterations, of which the first 3000 were burning iterations and then thinned to include only every second sample, thus for each estimated parameter 1000 distinct values are plotted. This plot was constructed using the `boxplot` function

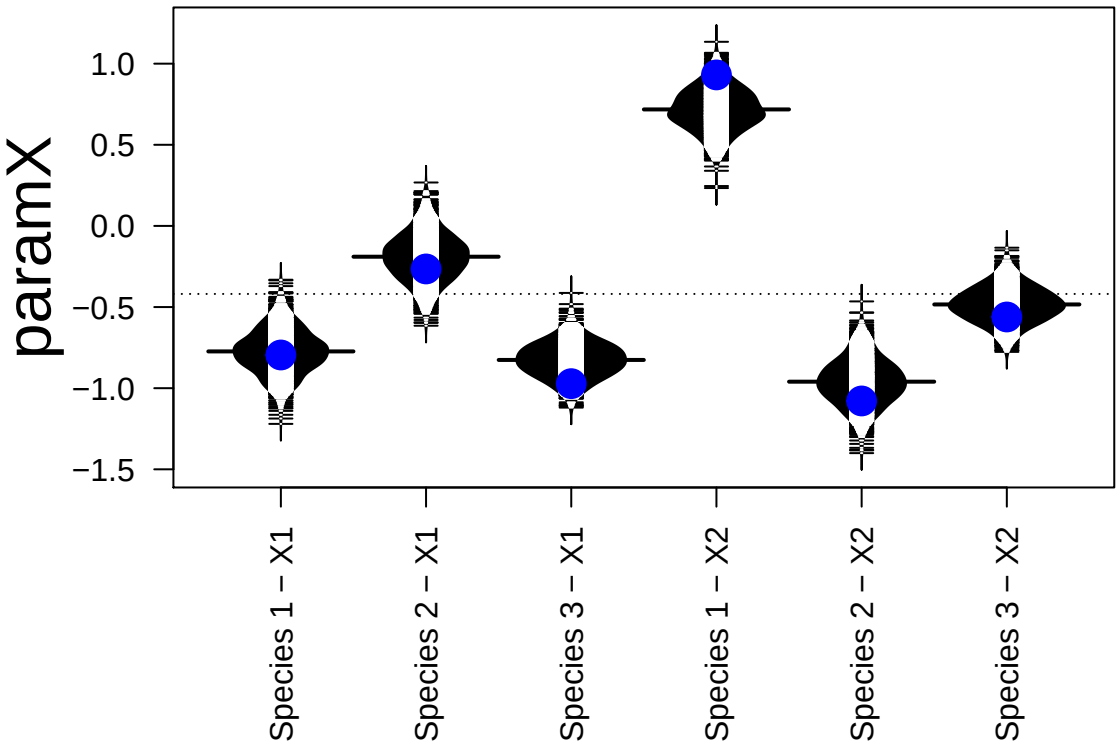

Figure 5: Violin plot showing the marginal distribution of the `paramX` parameters. The short segments on each violin plot represent the MCMC iterations for each parameters. The blue points correspond to the true parameter values (usually not known, but as this example is based on simulated data, they are known). The MCMC chain was ran for a total of 5000 iterations, of which the first 3000 were burning iterations and then thinned to include only every second sample, thus for each estimated parameter 1000 distinct values are plotted. This plot was constructed using `beanplot` of the `beanplot` package

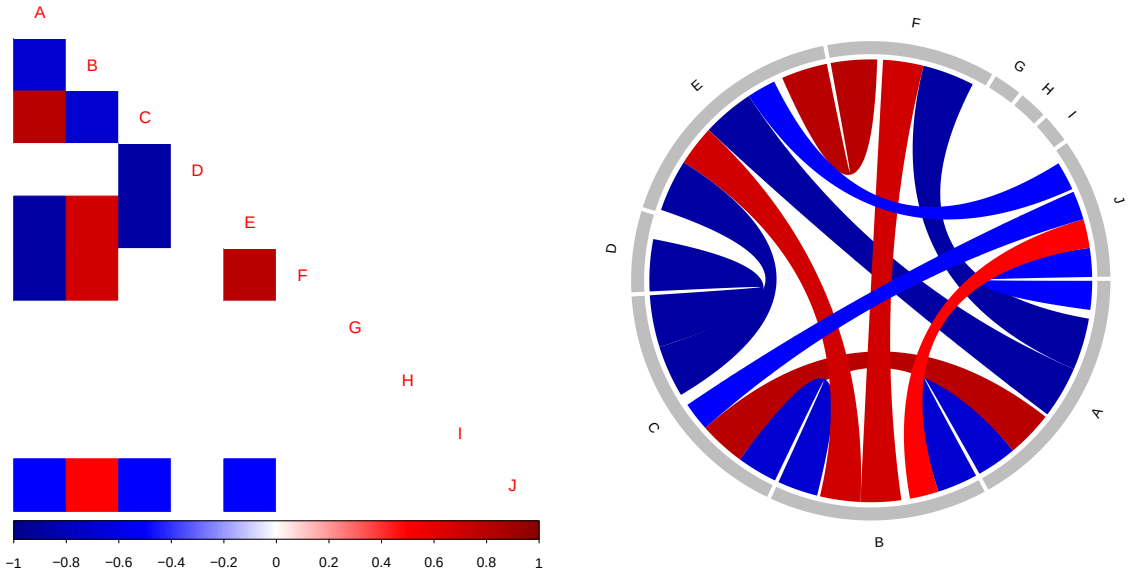

Figure 6: Matrix plot (left pannel) and chord diagram (right pannel) illustrating species-to-species associations measured by with correlation matrix **R**. Blue represents negative correlations while and red defines positive correlations between species pairs. Only those associations for which the upper and lower quantile of a given interval (given by the user, here a 95% credibility interval was used) which are either both positive or both negative are shown. It is only when the credibility interval does not overlap 0 that it is considered worth interpreting. The matrix plot was drawn using the `corrplot` function from `corrplot` package while the chord diagram was drawn using the `chordDiagram` function from `circlize` package. For the chord diagram, the width of grey bar defines the importance of this species with regard to the number of relation it has with other species.

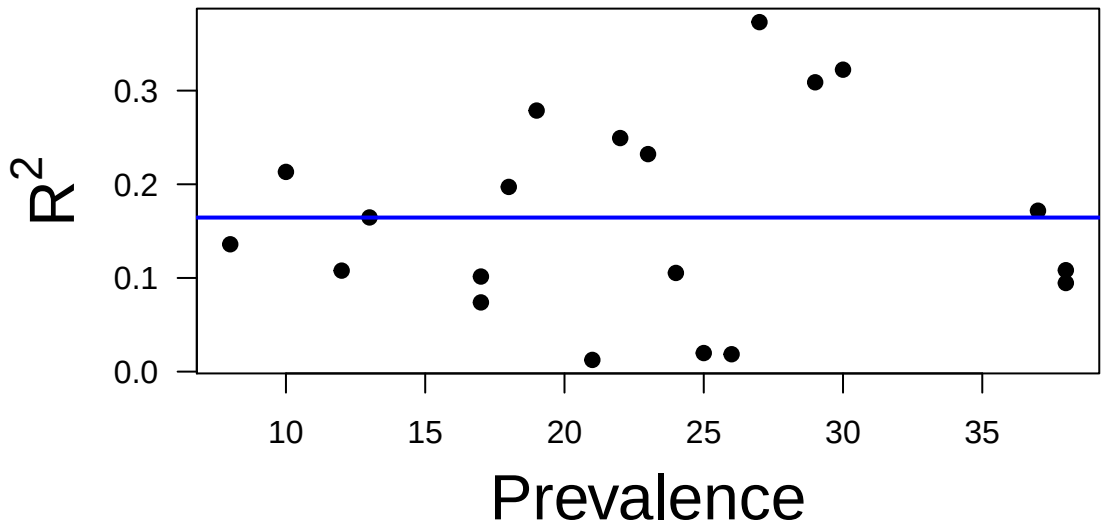

Figure 7: Examples of the R^2 summaries computed for binary (presence-absence) data. The dots correspond to the individual species. The species-specific R^2 values are plotted here against prevalence, i.e. fraction of occupied sampling units. The horizontal blue line give the mean values over the species, which can be used as an overall summary of the model's explanatory power. The code to generate this plot is detailed in the simulated example 1.

data, HMSC measures R^2 by calculating the proportion of explained variance over the total variance in the data [2]. For binary data, HMSC calculatesTjur's R^2 [3]. Tjur's R^2 is defined as the mean model prediction for those sampling units where the species occurs, minus the mean model prediction for those sampling units where the species does not occur. For abundance data, the predicted abundances are compared to the observed abundances by using Pearson's correlation. For binary data, when evaluating the model's predictions for higher hierarchical levels than the sampling unit, the sampling unit-level predictions are summed to generate predicted abundances, which are compared to the observed abundances by using Pearson's correlation. The upper limit of any of these R^2 measure is 1, which corresponds to the ideal case where the model completely replicates the data.

If the same data are used both to fit the model and evaluate its performance, the R^2 values may be inflated due to overfitting. Thus, while HMSC can produce R^2 summaries for the same data to which it is fitted, we recommend the user to split the data into separate training and validation data sets and compute an R^2 values for the validation data. We provided example scripts for how to separate training and validation data with the fungal and butterfly case studies. As illustrated in Figure 7, R^2 values can be computed separately for each species and then plotted against species' prevalence, abundance, or other such measures.

4.5 Variance partitioning

It is often of interest to partition the explained variation into components related to responses to (groups of) environmental covariates and to random effects, defined possibly at various levels. Furthermore, one can compute the amount of among-species variation in the responses of the species to the environmental covariates (informally, the variation among species niches), and ask which fraction of this can be explained by the species traits. The equations on which this variance partitioning is based are presented in the Supplementary Information of [1]. Figure 8 presents an example of variance partitioning.

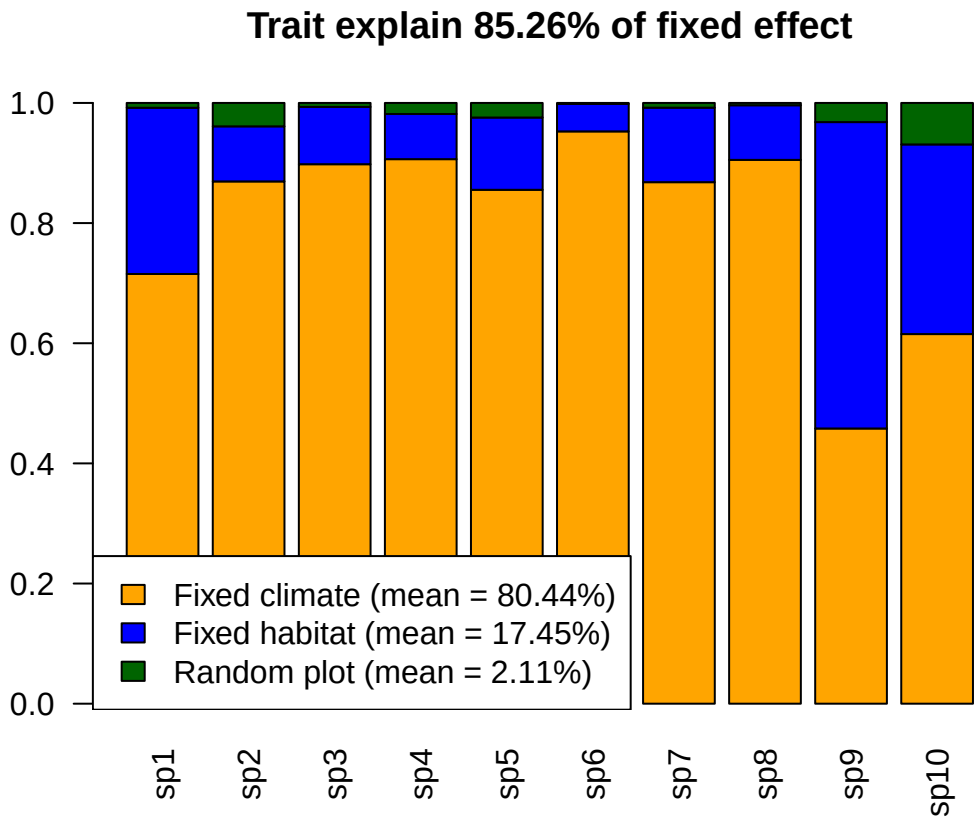

Figure 8: Example of a variance partitioning plot. Each bar corresponds to one species, and the colours shows the proportion of variance related to different variance components (the legend shows mean values over the species). As detailed in the examples, this plot can be generated using the function call plotVariancePartitioning, with HMSC-Matlab.

4.6 Predictions

The HMSC package offers the possibility of generating several kinds of predictions, or in other words, for generating simulated data. There are three kinds of choices related to generating predictions.

First, one can generate either expected values (e.g. occupancy probabilities) or actual realizations (e.g. presences or absences). The choice among these depends on the use of the prediction: if the interests is in, e.g., species-specific occurrence probabilities, it is more efficient to generate directly expected values rather than many realizations and then averaging over them. However, if the interest is in predicting co-occurrence patterns, one typically wishes to predict realizations (at least for new data, see below) , as the information about co-occurrence is not carried over with species-specific occupancy probabilities.

Second, one can generate predictions either for units that were included for estimation (training data) or for new units (called validation data or new data). Also intermediate cases are possible: e.g. to make a prediction for a new sampling unit present in a plot that was included in the training data.

Third, for new sampling units (i.e., sampling units that are not part of the data used for model fitting) it is possible to perform unconditional or conditional predictions. Unconditional predictions are the typical predictions needed to ask how the response variable (e.g. the occurrence of a species) changes under different environmental conditions (e.g. a changing value of an environmental covariate). When using conditional predictions, one can ask how inference on species occurrence is influenced by information about the occurrence of the same or other species in the same or other sampling units.

All types of predictions can be done for multiple samples of the model parameters drawn from the posterior distribution in order to propagate parameter uncertainty to the predictions. We illustrate various kinds of predictions with the simulated data examples and with the real examples related to fungi, butterflies, birds and bryophytes presented in the following sections.

5 Simulated Example 1

We start by illustrating how to perform analyses with the `HMSC` package with a simplified example consisting of presence-absence data for 10 species, acquired at 200 sampling units, belonging to 10 plots. Data for this example are found under the folder 'simulated' at `https://www.helsinki.fi/en/researchgroups/metapopulation-research-centre/hmsc` at the HMSC data link. The data that will be used for the analyses are in **.csv** format. At the moment three files in this folder will be used.

Y.csv Species occurrence data file. In this file, the species are in columns and sampling units in rows.

X.csv Environmental data file. In this file, the environmental covariates are in columns and sampling units in rows.

pi.csv Site and plot structure data file. In this file, the levels of the study design are in columns and sampling units are rows.

In this simulated example, we analysed these data with the aim of quantifying species niches (the responses of the species to the environmental covariates; through the parameters $\boldsymbol{\beta}$), as well as species-to-species association networks at the levels of sampling units and plots (through the parameters $\boldsymbol{\Omega}$).

Before moving on with the simulated example, let us make three remarks:

1. The data matrices do not need to be specifically in this format. Also file format other than .csv files can be used, and the files may or may not include headers. Furthermore, the information needed to construct e.g. the $\mathbf{Y}$ and $\mathbf{X}$ matrices can be in a single file or in two separate files. As long as the user can import the data files into R, great! In sub-section 5.3, we explain how to construct `HMSCdata` objects that are at the core of HMSC analyses.

2. In the Simulated Example 2 (section 6), we will return to these same data files, but we will additionally assume that we have information on species' traits ($\mathbf{T}$) and phylogenetic correlations among the species ($\mathbf{C}$). The reason for not including those yet is to keep this first example simple,

as well as to illustrate how additional layers can be added to the model if more data types are available.

5.1 Preliminaries

Before getting started on how to perform HMSC analyses, we will make the assumption that the user knows the basic of how to work within the R environment. As such, we will not explain in details what is a typical workflow in R.

5.2 Reading in the data

As a first step, lets read in the data files. The `HMSC` R packages does not provide any special functions for this, so the usual R-functions are to be used. Below, we construct the numerical data matrices as well as store the names of the species, covariates, and levels, so that we can relate to these when outputting the results.

```
# Community matrix
spComm <- read.csv("Y.csv")

# Environmental covariates
env <- read.csv("X.csv")

# Random effects
sitePlot <- read.csv("Pi.csv")
```

Note that after reading the site and plot structure (`Pi`), we converted each column into factors. This is a crucial step when constructing the `HMSCdata` object because site structure needs be considered as factor. The `HMSCdata` object contains all the basic information to construct the model.

5.3 Constructing an HMSC object in R

Formatting the data in such a way that it is of class `HMSCdata` is carried out using the `as.HMSCdata` function. Essentially, the `as.HMSCdata` function performs a series of test on the data to ensure it has the right format. When the data are in the right format and have passed the tests in the `as.HMSCdata` function, one can start the actual analyses.

It is common for the `as.HMSCdata` function to send various messages and warnings in the console querying the user to make sure that the data were formatted as the software requires. The `as.HMSCdata` function expects the data to be formatted in a particular way for the different types of data to be analysed (Figure 1). As such, error or warning messages may be sent for the users to organize the data in the proper format. For example, the species and environmental covariates should always contain numeric values while the random effects should always be a factor if there is only a single random effect, or a data frame if there are multiple random effects. If traits are considered in the analysis they should be composed only of numeric values while the phylogeny needs to be a square symmetric correlation or covariance matrix (if it is a covariance matrix, the `as.HMSCdata` function will convert it to a correlation matrix). As for the autocorrelated random effect, it has to be either a data frame with the first column being a factor while the other columns are spatial or temporal coordinates (the number of coordinates is not constrained) or a list of data frame following the structure presented previously if there are multiple autocorrelated random effects.

It is typically highly recommended to scale (center and divide by the standard deviation) the environmental covariates and the traits so that their mean is zero and their variance one. There are a few reasons for this, first this removes the potential effects units can have on the parameter estimation. Also, the default priors (discussed in sub-section 5.4) are compatible with the scaled covariates. In the `HMSCdata` function, the scaling can be done by setting to `TRUE` the logical arguments `scaleX` and `scaleTr`,

which respectively scale the environmental covariates and the traits. Note that the logical arguments `interceptX` and `interceptTr` automatically add an intercept to the environmental covariates and traits when set to `TRUE`. Because it is highly recommended to scale and add an intercept to environmental covariates and traits, the `scaleX`, `scaleTr`, `interceptX` and `interceptTr` are set to `TRUE` by default.

```
# Covert all columns of Pi to a factor

for(i in 1:ncol(sitePlot)){
    sitePlot[,i] <- as.factor(sitePlot[,i])
}

# Format data
simulEx1 <- as.HMSCdata(Y = spComm, X = env, Random = sitePlot,
                        interceptX = FALSE, scaleX = FALSE)

## [1] "column names were added to 'Random'"
## [1] "'Y' was converted to a matrix"
## [1] "'X' was converted to a matrix"
```

The `HMSC` R package includes this data, which can be accessed using the `data` function (see below). Note, however, that knowing how to convert any data into an object of class `HMSCdata` is the only way to go forward with the `HMSC` package. Therefore the previous steps are essential to read in any data into the `HMSC` R package. For simplicity, in the later examples we will import the data by loading it in the manner shown below.

```
data("simulEx1")
```

5.4 Defining the prior distributions

We next define the prior distributions through the `as.HMSCprior` function, which has an argument to define the parameters for each prior distribution used to construct the model. By default, the values used are the ones defined in the *Supporting information* of [1]. They are in essence, uniformative prior distributions. A user interested in modifying any prior distribution can do it by changing the argument associated to any specific parameters of the prior distribution. In the current and later examples, we will use the uniformative priors.

```
simulEx1prior <- as.HMSCprior(simulEx1)

## [1] "The priors for the latent variables should be OK for probit models but not necessarily for other
```

5.5 Setting the initial model parameters

In `HMSC`, if the user wants to initiate a specific set of parameters for the MCMC algorithm, it can be done with the `as.HMSCparam` function. Note that if only one or a reduced set of parameters needs to be specified, the `as.HMSCparam` function will automatically generate all the other parameters randomly.

```
simulEx1param <- as.HMSCparam(simulEx1, simulEx1prior)
```

5.6 Setting the values of the "true parameters"

As already mentioned, the true values of the parameters that structure the data are not usually known. Actually, if the true parameter values would be known, there would be no need for fitting the model and thus for estimating the parameters! But as the data we consider here have been generated by the same statistical model used to fit to the data, we know the true values of the parameters. We can thus compare the estimated parameter values to the true ones, and thus assess the performance of the model. With real data, this step is simply impossibles.

```
data("simulParamEx1")
```

5.7 Performing the MCMC sampling

In the context of a Bayesian analysis, model fitting or parameter estimation corresponds to the sampling of the posterior distribution. As is often done with hierarchical model structures, we utilize here Markov Chain Monte Carlo (MCMC) methods for posterior sampling. For a technically oriented user, by using conjugate prior distributions (or discrete priors for the spatial scale parameters and the phylogenetic signal parameter), we can utilize Gibbs sampling, i.e. sampling each parameter (which may be a scalar, vector or matrix) in turn directly from its full conditional distribution. As a result, there are no proposal-acceptance steps as is the case with e.g. the widely used Metropolis-Hastings algorithm. Thanks to the ability of sampling directly the full conditional distributions, the mixing of the parameters is generally rather good, and there is no need for tuning the sampling algorithm, i.e. automatic or manual adaptation. In spite of this, there are several parameters the user needs to set to guide the sampling process. However, the main take-home message for a user who is not technically oriented is that none of these choices influences the end result, i.e. the parameter estimates, assuming that the chains mix well enough (see the section on MCMC trace plots). Thus, the user may start her/his own analyses with the same settings that we use below, and then increase the amount of the iterations in the sampling if the mixing of the chains does not converge properly.

We are now ready to perform the MCMC sampling. In the code below we have decided to conduct the MCMC sampling for 10000 iterations, defined by the argument `niter`, with the first 1000 iterations are not considered in the model estimation (they are *burned*; destroyed), defined by the argument `nburn`. In addition, the MCMC run was thinned to remove the potential autocorrelation that might occur from sequential MCMC samples and to make the MCMC object more easily manageable. In the code below, thinning was set to 10 (`thin = 10`), which means that one in every 10 iterations was saved for model estimation.

When running the `hmsc` function, it is also essential to define the type of models that needs to be carried out. In the present example, since the community matrix describes the presence-absence of species at particular locations. a probit model was used, which was set by defining the argument `family = "probit"`.

It is recommended to carry out multiple MCMC runs, with different random initial values to make sure that the model converged properly.

In the `HMSC` package, all of these steps are set through the `hmsc` function, which is perform the MCMC sampling.

```
model <- hmsc(simulEx1, param = simulEx1param,
              priors = simulEx1prior, family = "probit",
              niter = 10000, nburn = 1000, thin = 10)

## iteration 2000
## iteration 4000
## iteration 6000
## iteration 8000
```

Note that the since it is rarely the case that parameters need to have a specific structure before starting the MCMC and that the uninformative priors as they are defined in [1] are a good starting points to construct an HMSC model, it is also possible to construct the model as above but without specifying explicitly the intitial parameters and the priors. As such, the code can be simplified to

```
model <- hmsc(simulEx1, family = "probit", niter = 10000,
              nburn = 1000, thin = 10)
```

5.8 Producing MCMC trace and density plots

The example codes below demonstrate how to generate MCMC trace and density plots for the different parameters (see Figures 2 and 3). Using the code below, objects of class `mcmc` are constructed from object of class `hmsc` from which trace and density plots can be drawn.

```
# Mixing objects
mixingParamX <- as.mcmc(model, parameters = "paramX")
mixingMeansParamX <- as.mcmc(model, parameters = "meansParamX")
mixingMeansVarX <- as.mcmc(model, parameters = "varX")
mixingParamLatent <- as.mcmc(model, parameters = "paramLatent")
```

As an example, we can draw trace and density plots for the `mixingMeansParamX` object. Note that in Figure 9 the `plot` function was used to draw both the trace and density plots together.

```
plot(mixingMeansParamX, col = "blue")
```

5.9 Producing posterior summaries

The folllowing example script shows how the marginal posterior distributions of the for the `mixingParamX` object can be summarized with violin plots (Figure 10), box plots (Figure 11), a table, and with credible intervals (Figure 12).

```
library(beanplot)

par(mar=c(6,4,1,1))
mixingParamXDF <- as.data.frame(mixingParamX)
beanplot(mixingParamXDF, las = 2)
points(1:30, as.vector(simulParamEx1$param$paramX), pch=19,
       col="blue", cex=2)
```

```
par(mar=c(6,4,1,1))
boxplot(mixingParamXDF, las = 2)
points(1:30, as.vector(simulParamEx1$param$paramX), pch=19,
       col="blue", cex=2)
```

```
# Average
average <- apply(model$results$estimation$paramX, 1:2, mean)

# 95% confidence intervals
```

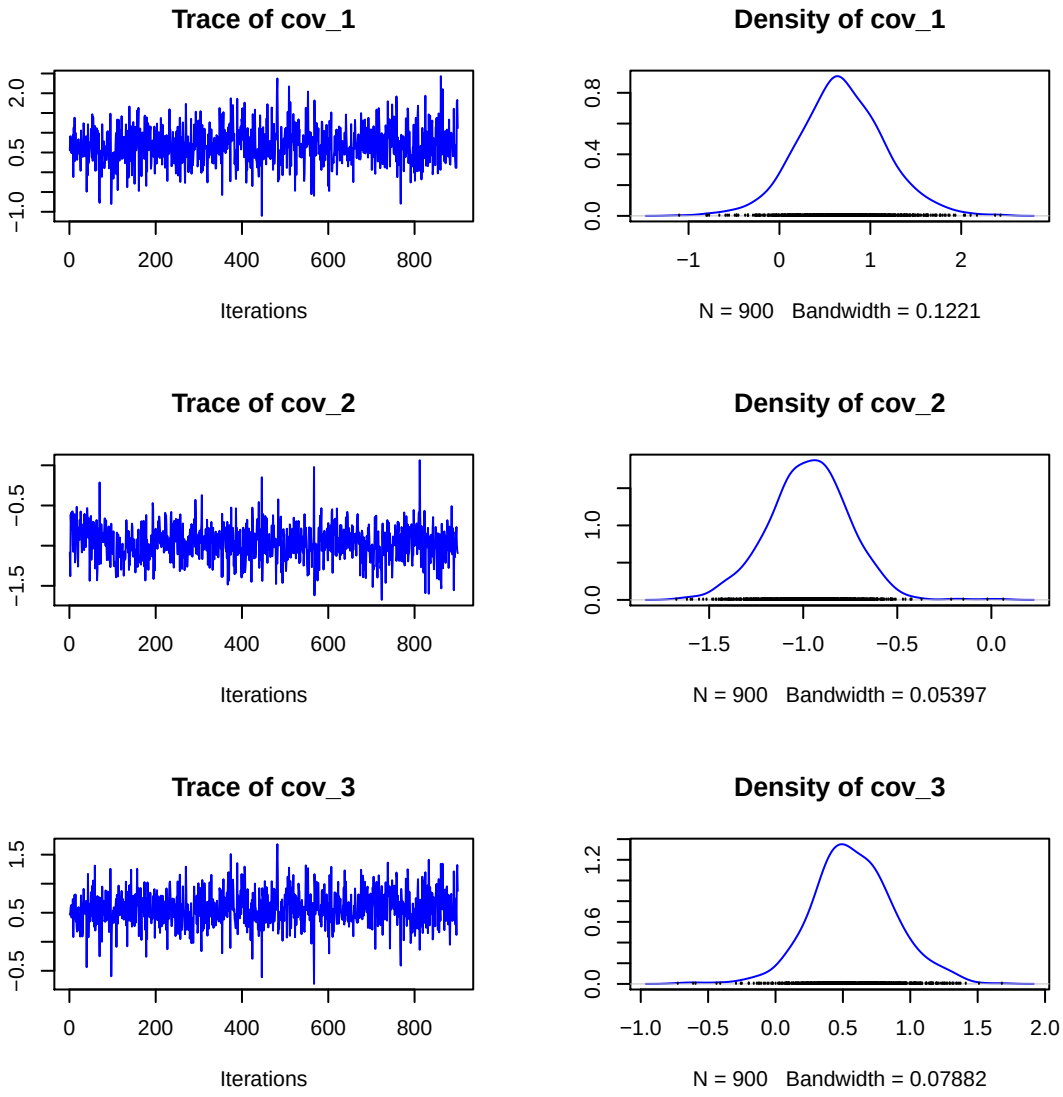

Figure 9: MCMC trace and density plot for the `mixingMeansParamX` object. This plot the barebone version of the trace and density plots as they present no graphical fluffing (except for the colour of the line... we could not resist!).

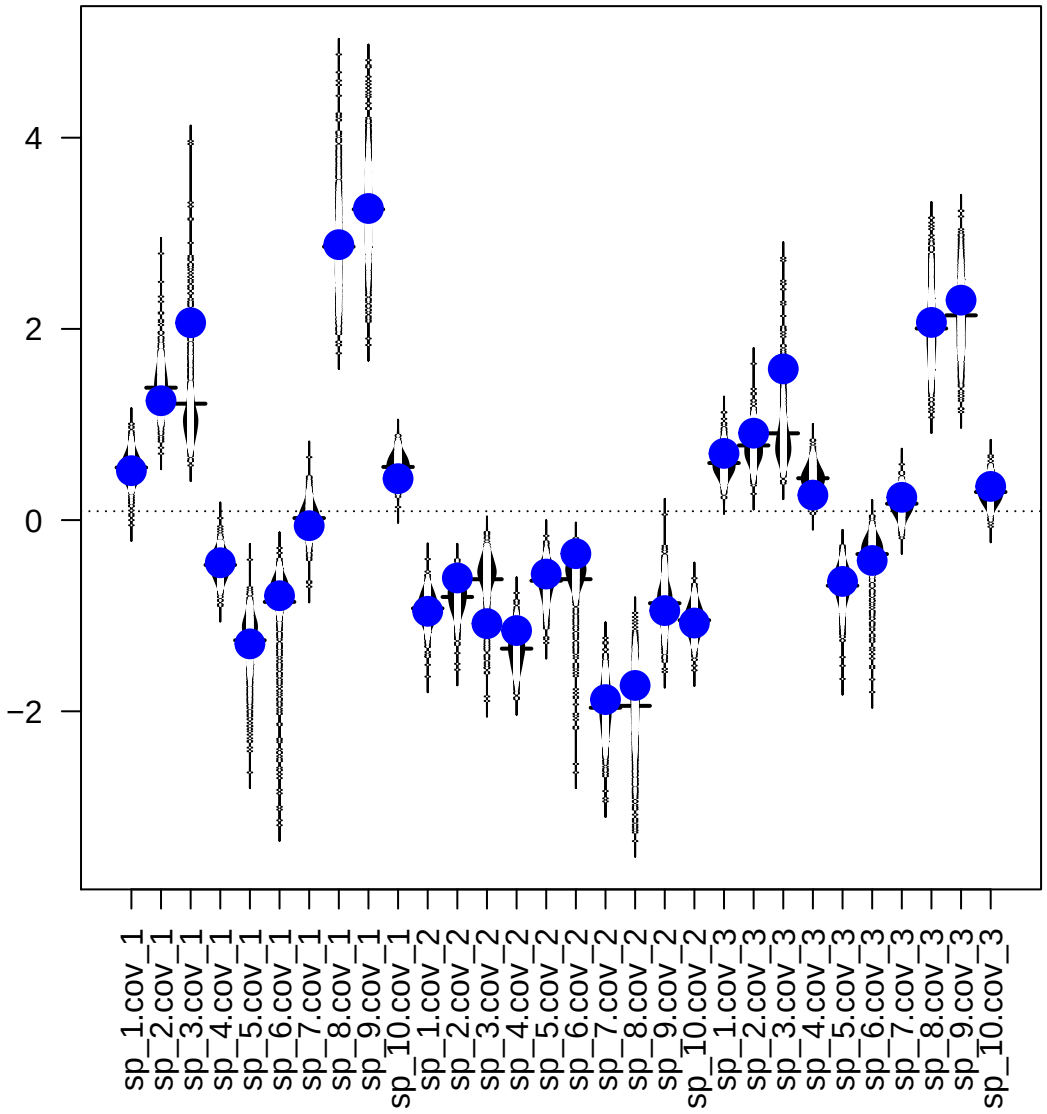

Figure 10: Violin plot showing the marginal distribution of the parameters in `paramX`. The short segments on each violin plot represent the MCMC iterations for each parameters. The blue points correspond to the true parameter values (usually not known, but as this example is based on simulated data, they are known). This plot was constructed using `beanplot` of the `beanplot` package.

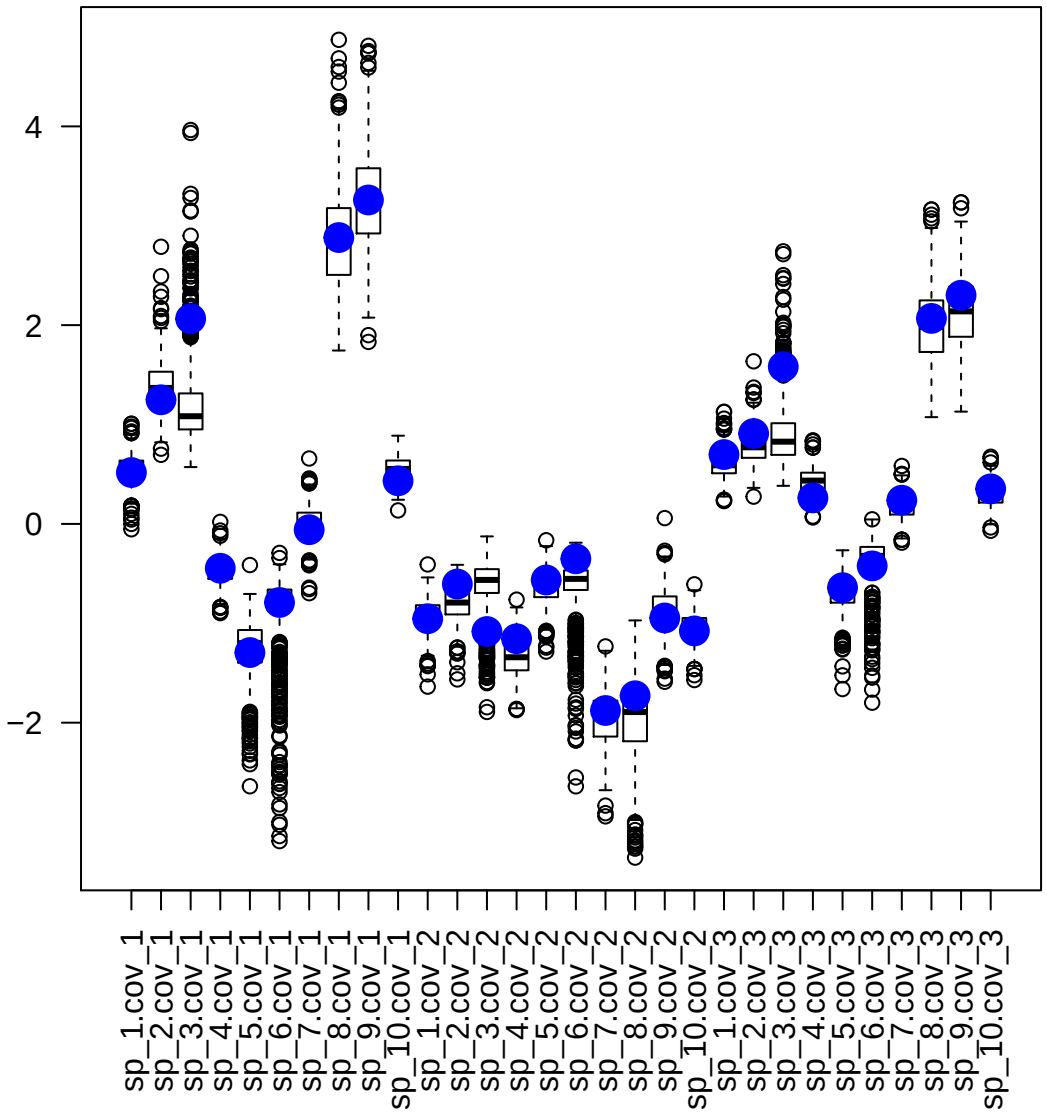

Figure 11: Box plot showing the marginal distribution of the parameters in `paramX`. The blue points correspond to the true parameter values (usually not known, but as this example is based on simulated data, they are known). This plot was constructed using the `boxplot` function.

```r
CI.025 <- apply(model$results$estimation$paramX, 1:2, quantile,
                probs = 0.025)
CI.975 <- apply(model$results$estimation$paramX, 1:2, quantile,
                probs = 0.975)
# Summary table
paramXCITable <- cbind(unlist(as.data.frame(average)),
                       unlist(as.data.frame(CI.025)),
                       unlist(as.data.frame(CI.975)))
colnames(paramXCITable) <- c("average", "lowerCI", "upperCI")
rownames(paramXCITable) <- paste(rep(colnames(average),
                                     each = nrow(average)), "_",
                                     rep(rownames(average),
                                         ncol(average)), sep="")

# Print summary table
paramXCITable
```

```
##                  average     lowerCI     upperCI
## cov_1_sp_1   0.54959799  0.29775450  0.84130528
## cov_1_sp_2   1.38508612  0.99784758  1.84750912
## cov_1_sp_3   1.21780804  0.73065013  2.50120068
## cov_1_sp_4  -0.46912198 -0.72140303 -0.22794366
## cov_1_sp_5  -1.25698301 -1.95899584 -0.85423095
## cov_1_sp_6  -0.85537094 -2.29428083 -0.48647344
## cov_1_sp_7   0.02104182 -0.27128000  0.33977411
## cov_1_sp_8   2.85976719  2.02707056  3.96503272
## cov_1_sp_9   3.25083198  2.36043502  4.13604809
## cov_1_sp_10  0.55593071  0.34308652  0.79370757
## cov_2_sp_1  -0.92303665 -1.23526262 -0.65601125
## cov_2_sp_2  -0.80379238 -1.16224355 -0.51747872
## cov_2_sp_3  -0.61836508 -1.33763999 -0.28931507
## cov_2_sp_4  -1.34445158 -1.70198636 -0.99659749
## cov_2_sp_5  -0.63398886 -1.02574799 -0.34216049
## cov_2_sp_6  -0.61767910 -1.51239968 -0.31138957
## cov_2_sp_7  -1.96268934 -2.46729386 -1.48577719
## cov_2_sp_8  -1.94318302 -2.87596812 -1.27770778
## cov_2_sp_9  -0.86937298 -1.33168398 -0.41700400
## cov_2_sp_10 -1.04737526 -1.33132891 -0.76453298
## cov_3_sp_1   0.59746612  0.34493562  0.86939893
## cov_3_sp_2   0.78106225  0.47195414  1.10924045
## cov_3_sp_3   0.90787446  0.51572090  1.74225272
## cov_3_sp_4   0.43793550  0.19120907  0.68015116
## cov_3_sp_5  -0.68536298 -1.11743702 -0.37220204
## cov_3_sp_6  -0.35544770 -1.03086843 -0.07799246
## cov_3_sp_7   0.17189710 -0.08336199  0.39424644
## cov_3_sp_8   2.00451345  1.35548616  2.76913672
## cov_3_sp_9   2.14184849  1.48621352  2.83300478
## cov_3_sp_10  0.29406987  0.08625814  0.51502414
```

```r
par(mar=c(7,4,1,1))
```

```
plot(0, 0, xlim = c(1, nrow(paramXCITable)),
     ylim = range(paramXCITable), type = "n",
     xlab = "", ylab = "", main="paramX", xaxt="n")

axis(1,1:30,rownames(paramXCITable),las=2)

abline(h = 0,col = "grey")
arrows(x0 = 1:nrow(paramXCITable), x1 = 1:nrow(paramXCITable),
       y0 = paramXCITable[, 2], y1 = paramXCITable[, 3],
       code = 3, angle = 90, length = 0.05)

points(1:nrow(paramXCITable), paramXCITable[,1], pch = 15,
       cex = 2)

points(1:nrow(paramXCITable),
       as.vector(simulParamEx1$param$paramX),
       col = "blue", pch = 19, cex = 2)
```

5.10 Variance partitioning

With the code below the user can perform variance partitioning for the model. By using the `barplot` function, a graphic such as the one presented in Figure 13 can readily be drawn.

```
variationPart <- variPart(model, c(rep("climate", 2), "habitat"))
variationPart
```

```
##         climate    habitat     random1     random2
## sp_1  0.6348616 0.27111998 0.067708285 0.026310139
## sp_2  0.4619460 0.43651268 0.052642170 0.048899146
## sp_3  0.2323109 0.48933177 0.264112619 0.014244745
## sp_4  0.8799578 0.10011273 0.012331301 0.007598178
## sp_5  0.3219639 0.37181092 0.280103118 0.026122072
## sp_6  0.4327187 0.15255112 0.380301946 0.034428218
## sp_7  0.9708202 0.01148919 0.005187866 0.012502710
## sp_8  0.4787863 0.51341231 0.002963940 0.004837467
## sp_9  0.1447542 0.84183443 0.004740562 0.008670796
## sp_10 0.8845994 0.07861161 0.026906252 0.009882757
```

```
Colour <- c("orange", "blue", "darkgreen", "purple")

barplot(t(variationPart), col=Colour, las=1)

legend("bottomleft",
       legend = c(paste("Fixed climate (mean = ",
                        round(mean(variationPart[, 1]), 4)*100, "%)",
                        sep=""),
                  paste("Fixed habitat (mean = ",
                        round(mean(variationPart[, 2]), 4)*100, "%)",
                        sep=""),
                  paste("Random site (mean = ",
```

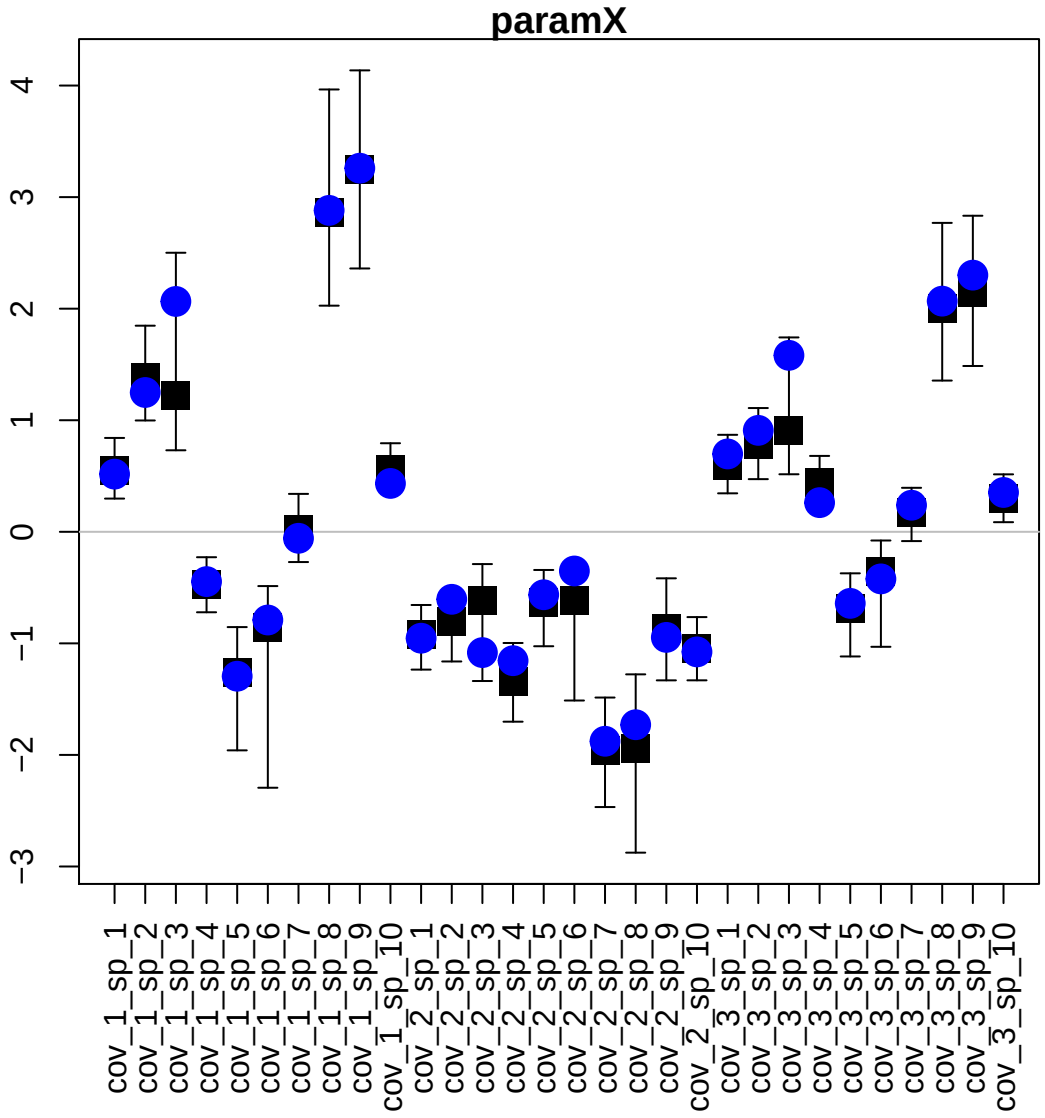

Figure 12: Credibility interval describing the marginal distribution for the parameters in `paramX`. The blue points correspond to the true parameter values (usually not known, but as this example is based on simulated data, they are known).

```
                        round(mean(variationPart[, 3]), 4)*100, "%)",
                        sep=""),
                  paste("Random plot (mean = ",
                        round(mean(variationPart[, 4]), 4)*100, "%)",
                        sep="")),
       fill = Colour, bg="white")
```

5.11 Association networks

The code below shows how to plot species-to-species association networks using chord diagrams and matrix plot for site-level (Figures 14) and the plot-level (Figures 15) random effect. As can be seen in the code, correlations, which are bounded between -1 and 1, were used to more clearly illustrate the relationships between pairs of species.

```
# Extract all estimated associatin matrix
assoMat <- corRandomEff(model)

# Average
siteMean <- apply(assoMat[, , , 1], 1:2, mean)
plotMean <- apply(assoMat[, , , 2], 1:2, mean)

#=======================
### Associations to draw
#=======================
#--------------------
### Site level effect
#--------------------
# Build matrix of colours for chordDiagram
siteDrawCol <- matrix(NA, nrow = nrow(siteMean),
                      ncol = ncol(siteMean))

siteDrawCol[which(siteMean > 0.4, arr.ind=TRUE)]<-"red"
siteDrawCol[which(siteMean < -0.4, arr.ind=TRUE)]<-"blue"

# Build matrix of "significance" for corrplot
siteDraw <- siteDrawCol
siteDraw[which(!is.na(siteDraw), arr.ind = TRUE)] <- 0
siteDraw[which(is.na(siteDraw), arr.ind = TRUE)] <- 1
siteDraw <- matrix(as.numeric(siteDraw), nrow = nrow(siteMean),
                   ncol = ncol(siteMean))

#--------------------
### Plot level effect
#--------------------
# Build matrix of colours for chordDiagram
plotDrawCol <- matrix(NA, nrow = nrow(plotMean),
                      ncol = ncol(plotMean))

plotDrawCol[which(plotMean > 0.4, arr.ind=TRUE)]<-"red"
plotDrawCol[which(plotMean < -0.4, arr.ind=TRUE)]<-"blue"
```

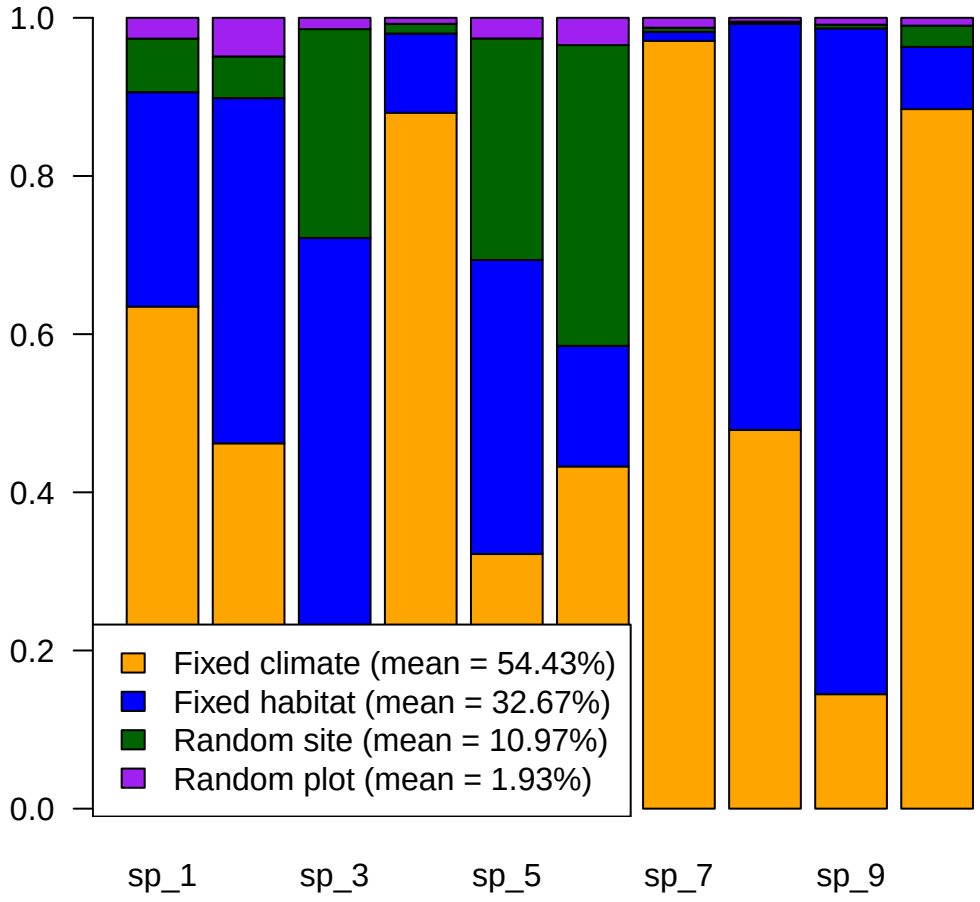

Figure 13: Variance partitioning results for simulated example 1. It is important to note here that what is partitioned is the **explained** variance of the model.

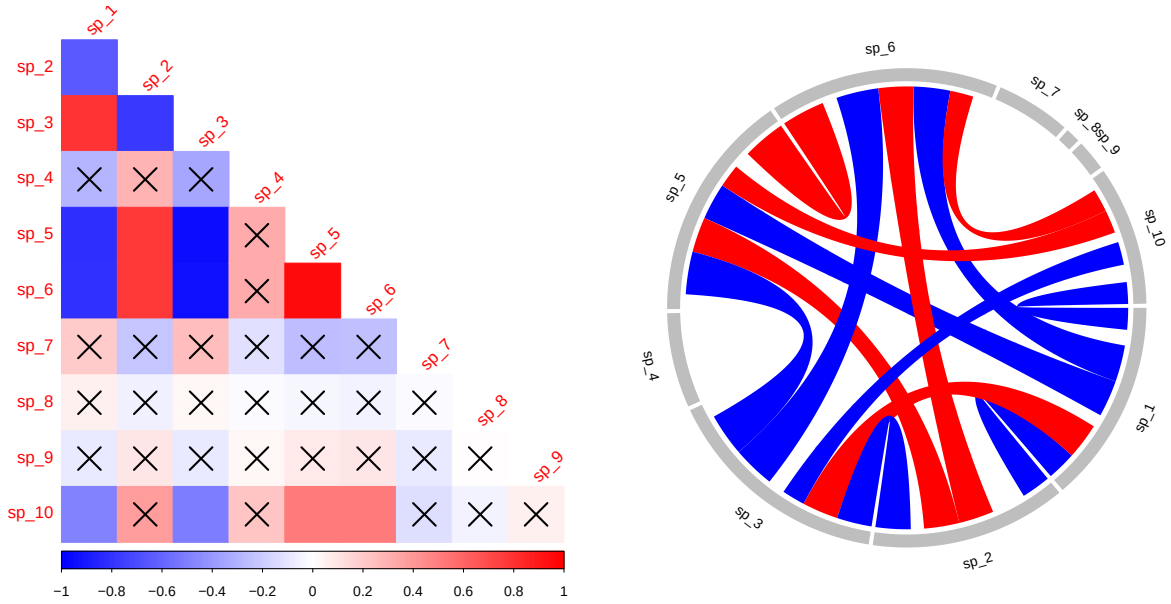

Figure 14: Association network presented for the site-level random effect for simulated example 1. The left panel is in the form of a matrix plot (drawn using the `corrplot` function of the `corrplot` package) while the right panel the association network is drawn with a chord diagram (using the `chordDiagram` function of the `circlize` package). In the matrix plot, the cells mark with an × present an absolute correlation value smaller than 0.4. Similarly, in the chord diagram only the correlation with absolute values larger than 0.4 were drawn.

```
# Build matrix of "significance" for corrplot
plotDraw <- plotDrawCol
plotDraw[which(!is.na(plotDraw), arr.ind = TRUE)] <- 0
plotDraw[which(is.na(plotDraw), arr.ind = TRUE)] <- 1
plotDraw <- matrix(as.numeric(plotDraw), nrow = nrow(plotMean),
                   ncol = ncol(plotMean))
```

```
library(corrplot)
par(mfrow=c(1,2))
# Matrix plot
Colour <- colorRampPalette(c("blue", "white", "red"))(200)
corrplot(siteMean, method = "color", col = Colour, type = "lower",
         diag = FALSE, p.mat = siteDraw, tl.srt = 45)

# Chord diagram
chordDiagram(siteMean, symmetric = TRUE,
             annotationTrack = c("name", "grid"),
             grid.col = "grey", col = siteDrawCol)
```

```
par(mfrow=c(1,2))
# Matrix plot
Colour <- colorRampPalette(c("blue", "white", "red"))(200)
```

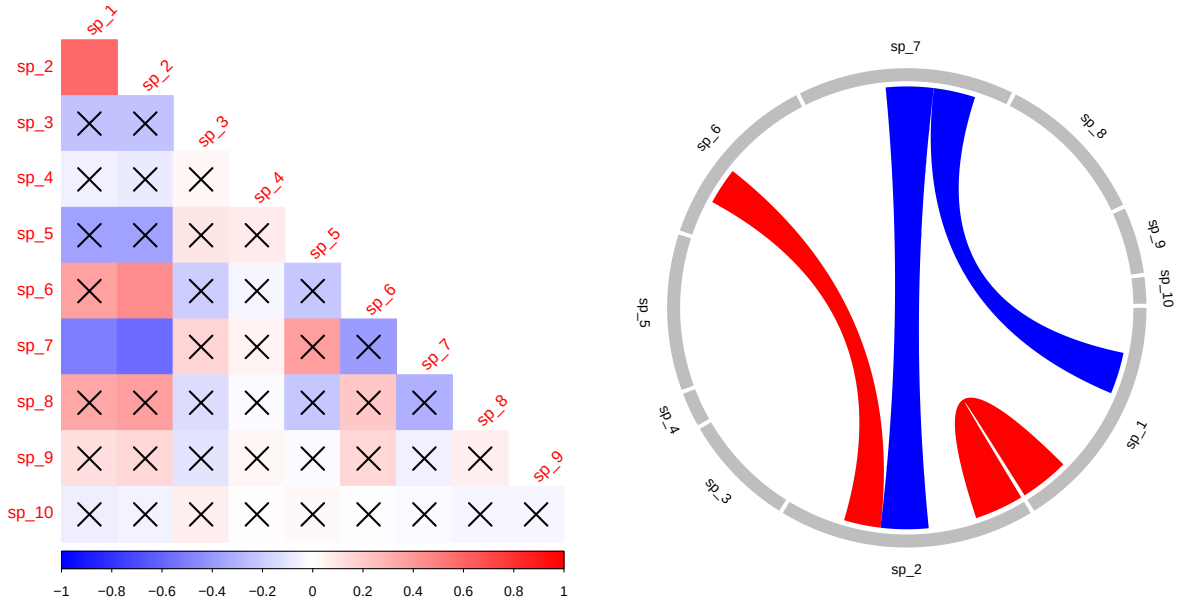

Figure 15: Association network presented for the plot-level random effect for simulated example 1. The left panel is in the form of a matrix plot (drawn using the `corrplot` function of the `corrplot` package) while the right panel the association network is drawn with a chord diagram (using the `chordDiagram` function of the `circlize` package). In the matrix plot, the cells mark with an × present an absolute correlation value smaller than 0.4. Similarly, in the chord diagram only the correlation with absolute values larger than 0.4 were drawn.

```
corrplot(plotMean, method = "color", col = Colour, type = "lower",
         diag = FALSE, p.mat = plotDraw, tl.srt = 45)

# Chord diagram
chordDiagram(plotMean, symmetric = TRUE,
             annotationTrack = c("name", "grid"),
             grid.col = "grey", col = plotDrawCol)
```

5.12 Computing the explanatory power of the model

With the example scripts below the user can compute coefficient of determinations (R^2), which quantify the explanatory power of the model at each of the levels of the study design. In this case, the R^2 values are computed for the site-level and plot-levels random effect.

```
# Prevalence
prevSp <- colSums(simulEx1$Y)

# Coefficient of multiple determination
R2 <- Rsquared(model, averageSp = FALSE)
R2comm <- Rsquared(model, averageSp = TRUE)

# Draw figure
par(mar=c(5,6,0.5,0.5))
```

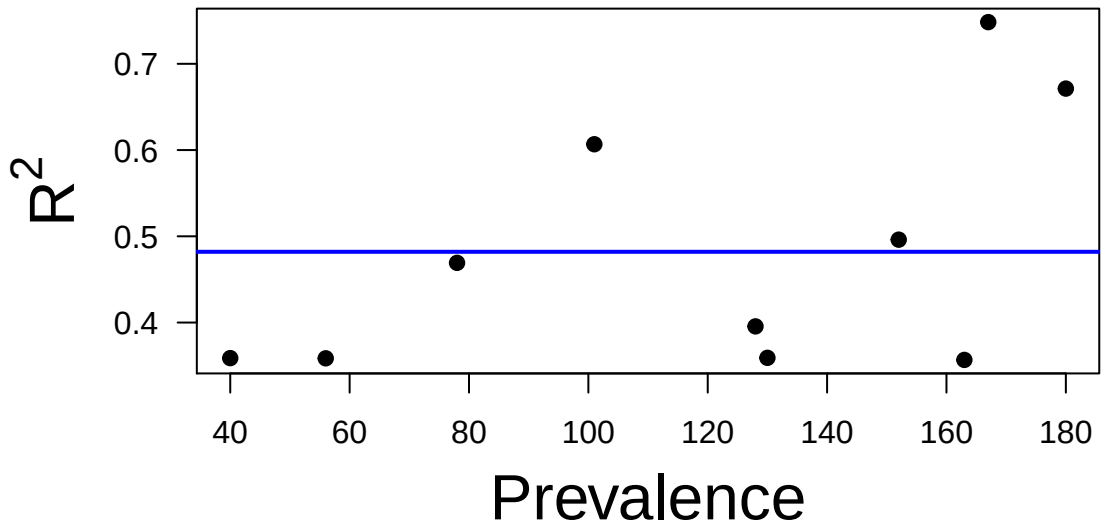

Figure 16: R^2 summaries computed for binary (presence-absence) data. The dots correspond to the individual species. The species-specific R^2 values are plotted here against prevalence, i.e. fraction of occupied sampling units. The horizontal blue line give the mean values over the species, which can be used as an overall summary of the model's explanatory power.

```
plot(prevSp, R2, xlab = "Prevalence",
     ylab = expression(R^2), pch=19, las=1,cex.lab = 2)
abline(h = R2comm, col = "blue", lwd = 2)
```

5.13 Sampling the posterior distribution

As illustrated by the code below, the user can access the all the detailed MCMC run for each parameter and can calculate the full joint posterior distribution to do any kind of post-processing of the results beyond what is provided by the specific tools offered in the `HMSC` package. It is recommended not to run the first line of the code in the next code chunck directly because it is irrelevant to visualize all values of the MCMC run on screen without any summarization.

```
# Extract all MCMC of paramX
model$results$estimation$paramX

### Full joint probability distribution
fullPost <- jposterior(model)
```

5.14 Generating predictions for training data

As mentionned in section 5.13, `HMSC` can be used to generate many kinds of predictions. Here we illustrate the predictions that are *not* conditional on the occurrences of other species.

We first make predictions for the training data, i.e. for the model we fitted that include the same environmental conditions (as described in $\mathbf{X}$) and the site- and plot-level random effect (as described in the matrix $\mathbf{\Pi}$). The motivation for computing such predictions may be, e.g., to evaluate the quality of

fitted model. To illustrate this, we use the predictions to compute the species-specific Tjur's R^2 values (for which the `Rsquared` function can be used, as we illustrated in section 5.12).

Since in the present case we are interested in species occurrence probabilities, we predicted expectations (occurrence probabilities) rather than realizations (zeros or ones). To account for parameter uncertainty, the `predict` function generate the predictions for all the posterior samples (excluding burned and thinned iterations).

```
predTrain <- predict(model)
```

5.15 Generating predictions for new data

Making predictions for new data (i.e. data outside the training data) are commonly needed for at least two kinds of purposes. First, they are needed to compare the model predictions with validation data, and thus examine the predictive power of the model more thoroughly than the approach proposed in section 5.14. Second, such predictions are needed in a simulation context, e.g. to ask how species occurrence depends on environmental conditions. In the current simulated example, the design matrix involves the intercept and two environmental covariates. If we assume that we are interested in how the species occurrences depend on variation of the second environmental covariate, we can construct a new matrix $\mathbf{X}$ representing environmental conditions where the values of the first covariate are fixed to its mean, and the second covariate varies systematically to cover the range of interest. As we do not want to apply the random effects estimated for the specific plots and sites included in the training data, we also specify that the units for which the predictions are to be generated are new, so we just assign new units (which are alphanumerically different from which were defined during model fitting) to the new $\mathbf{\Pi}$-matrix.

Note that when organizing the new data to be used for validation, it ultimately has to be structured into an object of class `HMSCdata`, which means that it has to be pass through the `as.HMSCdata` function.

```
# New environmental covariates
newPred <- 100
nEnv <- ncol(simulEx1$X)
Xnew <- matrix(nrow = newPred, ncol = nEnv)
colnames(Xnew) <- colnames(simulEx1$X)

Xnew[, 1] <- 1
Xnew[, 2] <- mean(simulEx1$X[, 2])
Xnew[, 3] <- seq(min(simulEx1$X[, 3]), max(simulEx1$X[, 3]),
                 length = newPred)

# New site- and plot-level random effect
RandomSel <- sample(200, 100)
RandomNew <- simulEx1$Random[RandomSel, ]
for(i in 1:ncol(RandomNew)){
        RandomNew[, i] <- as.factor(as.character(RandomNew[, i]))
}
colnames(RandomNew) <- colnames(simulEx1$Random)

# Organize the data into an HMSCdata object
dataVal <- as.HMSCdata(X = Xnew, Random = RandomNew,
                       scaleX = FALSE, interceptX = FALSE)

## [1] "column names were added to 'Random'"
## [1] "row names were added to 'X'"
```

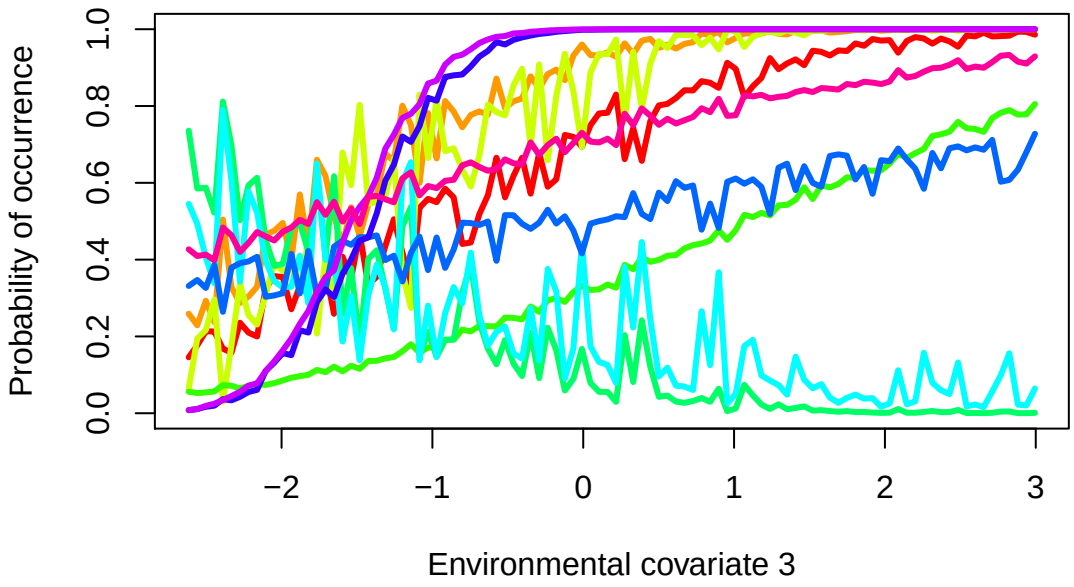

Figure 17: Predicted responses to covariate 3, when covariates 1 (intercept) and 2 (the first measured covariate) are set to their mean values. The different colours indicate different species.

```
predVal <- predict(model, newdata = dataVal)
```

After performing this calculations, we are ready to plot the prediction. For example, we can plot how the species respond to variation in the environmental covariate of interest (illustrated in Figure 17).

```
plot(0, 0, type="n", xlim = range(dataVal$X[, 3]), ylim = c(0, 1),
     xlab = "Environmental covariate 3",
     ylab = "Probability of occurrence")

Colours <- rainbow(ncol(predVal))
for(i in 1:ncol(predVal)){
  lines(dataVal$X[, 3], predVal[, i], col = Colours[i], lwd=3)
}
```

6 Simulated example 2

We next continue the HMSC analyses by assuming that, in addition to the occurrence data and the environmental covariate data, we have access to species' trait data and phylogenetic correlations. The data for this example are found under the folder 'simulated' at `https://www.helsinki.fi/en/researchgroups/metapopulation-research-centre/hmsc` at the HMSC data link. In addition of the **Y.csv**, **X.csv** and **pi.csv** files, which we have analysed in the simulated example 1 (section 5), in simulated example 2, we

consider also the **T.csv** and **C.csv** files which contain the trait matrix **T** and the phylogenetic correlation matrix **C**.

We will now analyse this extended data set to address not only the same questions as in simulated example 1, but also two additional questions. First, with this extend data set we can examine how traits influence species niches and thus estimate the parameters $\boldsymbol{\gamma}$. Second, we can ask if species niches show a phylogenetic signal after accounting to the variation explained by the traits, and thus estimate parameter ρ.

As many parts of the script for simulated example 1 are identical to the script in simulated example 2, we explain below only the parts related to traits and phylogeny. Thus a user going exclusively through this code can also check the explanations given for simulated example 1 (section 5).

6.1 Organizing the data

In addition to reading from the file the same data as for simulated example 1 (section 5), we need to read the traits and phylogenetic correlation matrices. In the code below, we also store the names of the traits so that we can relate to them later.

To make sure there are no confusions, here is how the different parts of the data need to be organized to be used in the `HMSC` package.

```
# Community matrix
spComm <- read.csv("Y.csv")

# Environmental covariates
env <- read.csv("X.csv")

# Random effects
sitePlot <- read.csv("Pi.csv")

# Traits
traits <- read.csv("T.csv")

# Phylogeny
phylo <- read.csv("C.csv")
```

```
# Covert all columns of Pi to a factor
for(i in 1:ncol(sitePlot)){
    sitePlot[,i] <- as.factor(sitePlot[,i])
}

# Format data
rownames(phylo) <- colnames(phylo)
simulEx2 <- as.HMSCdata(Y = spComm, X = env, Tr = t(traits),
                        Phylo = phylo, Random = sitePlot,
                        interceptX = FALSE, scaleX = FALSE)
```

Recall that the object `phylo` needs to be square symmetric matrix and as such the `as.HMSCdata` function will also require that the rows and column names match.

As for the simulated example 1 (section 5), these data are already available in the `HMSC` package with the true model parameters.

```
data(simulEx2)
data(simulParamEx2)
```

6.2 Additional output from a model with traits and phylogenetic correlations

After estimating the parameters for the model using the data for simulated example 2 using the code below, we can start studying the parameters related to traits and phylogeny.

```
model <- hmsc(simulEx2, family = "probit", niter = 10000,
              nburn = 1000, thin = 10)

## [1] "The priors for the latent variables should be OK for probit models but not necessarily for other
## iteration 2000
## iteration 4000
## iteration 6000
## iteration 8000
```

In addition of the results already introduced for simulated example 1 (section 5), we can now examine the estimated parameters related to traits and phylogeny.

The code below shows how to access the posterior distribution of the parameters $\boldsymbol{\gamma}$ (`paramTr`) and ρ (`paramPhylo`), how to produce MCMC trace and density plots showing the mixing of the parameter, as well as how to produce various other posterior summaries.

The following code chunks show how to obtain posterior summaries of the parameters associated to traits. First, trace and density plots were drawn for each paramters in `paramTr` (Figure 18), following violin plots are drawn to study the marginal distribution of the paramters (Figure 19).

```
### Mixing object
mixing <- as.mcmc(model, parameters = "paramTr")

### Draw trace and density plots for all combination of paramters
par(mfrow=c(6,2))
plot(mixing, col = "blue", auto.layout = FALSE)
```

```
### Mixing object
mixingDF <- as.data.frame(mixing)

### Draw trace and density plots for all combination of paramters
beanplot(mixingDF, las = 2)
points(1:ncol(mixingDF),as.vector(simulParamEx2$param$paramTr),
       col = "blue", pch = 19, cex = 2)
```

We can also draw similar plots to study phylogeny. As for the previous code chunks, first a trace and a density plot were drawn for `paramPhylo` (Figure 20), following a box plot was drawn to study the marginal distribution of the paramter (Figure 21).

```
### Mixing object
mixing <- as.mcmc(model, parameters = "paramPhylo")

### Draw trace and density plots for all combination of paramters
plot(mixing, col = "blue")
```

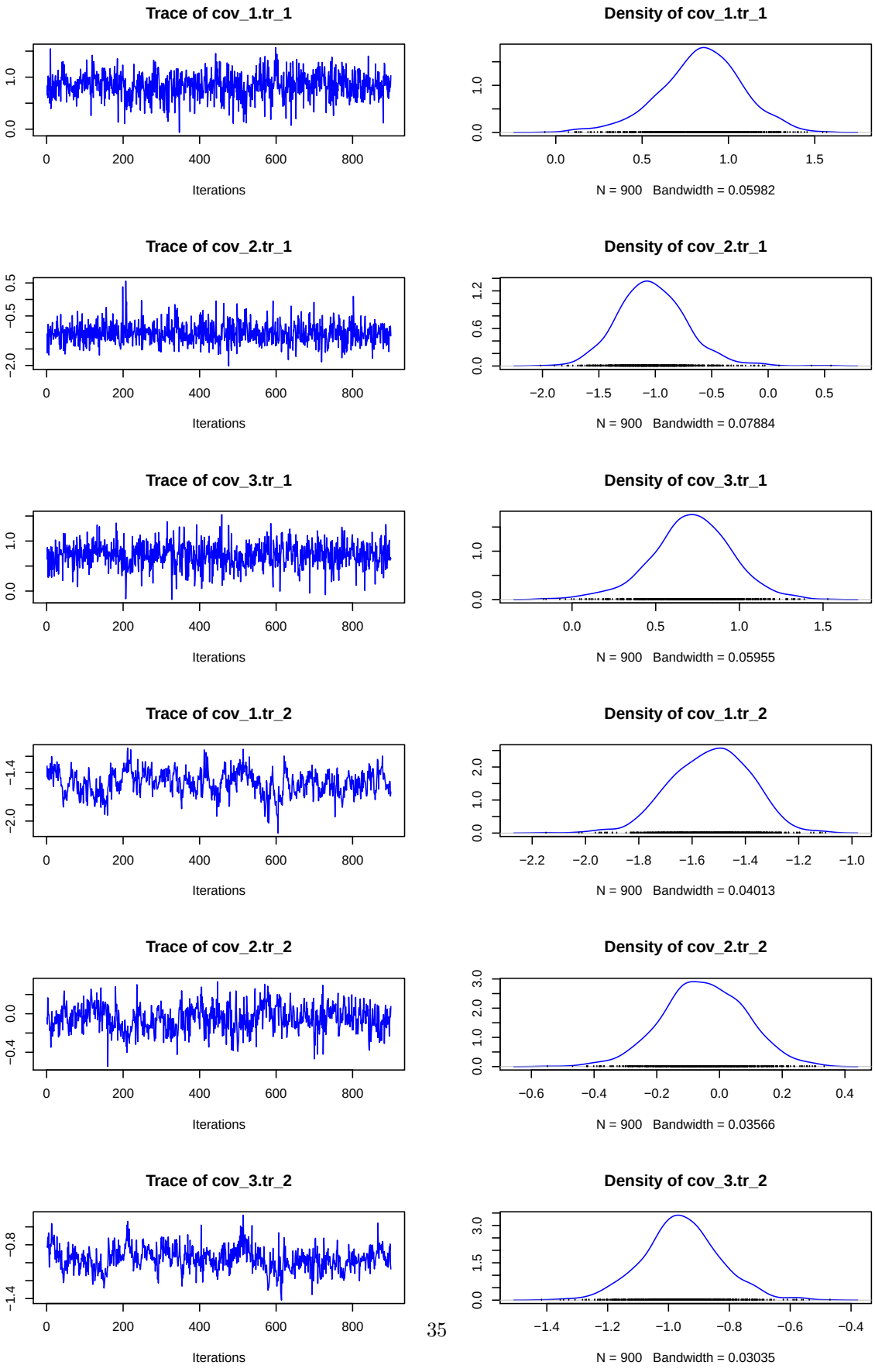

Figure 18: MCMC trace and density plots for `paramTr`.

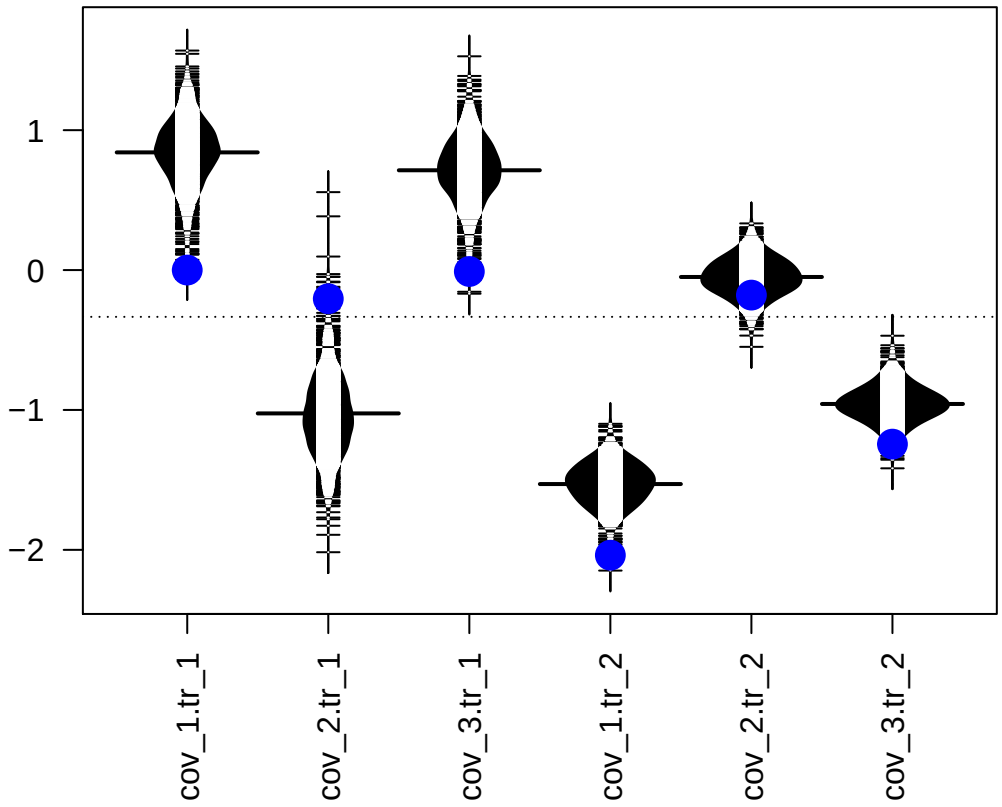

Figure 19: Violin plot showing the marginal distribution of the parameters in `paramTr`. The short segments on each violin plot represent the MCMC iterations for each parameters. The blue points correspond to the true parameter values (usually not known, but as this example is based on simulated data, they are known). This plot was constructed using `beanplot` of the `beanplot` package.

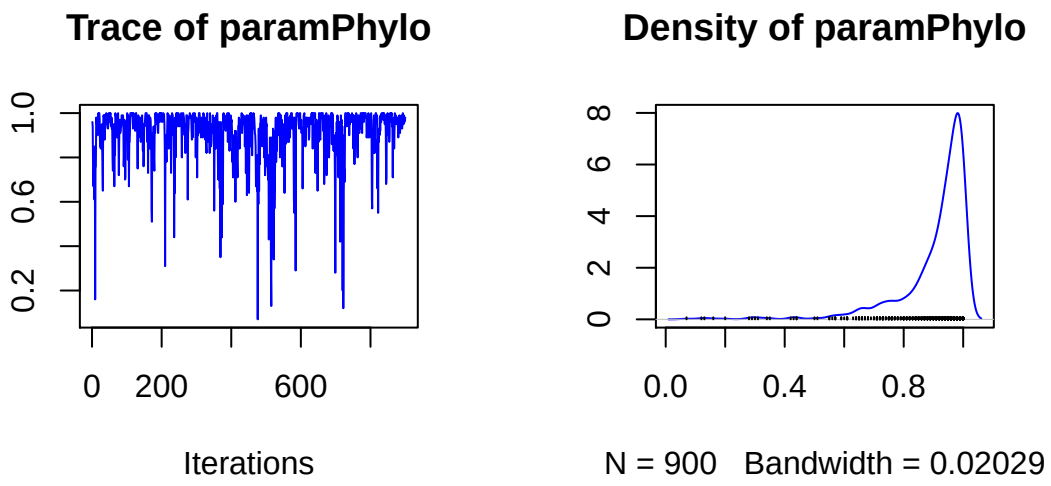

Figure 20: MCMC trace and density plot for `paramPhylo`.

```
### Mixing object
mixingDF <- as.data.frame(mixing)

### Draw trace and density plots for all combination of paramters
boxplot(mixingDF, las = 2)
```

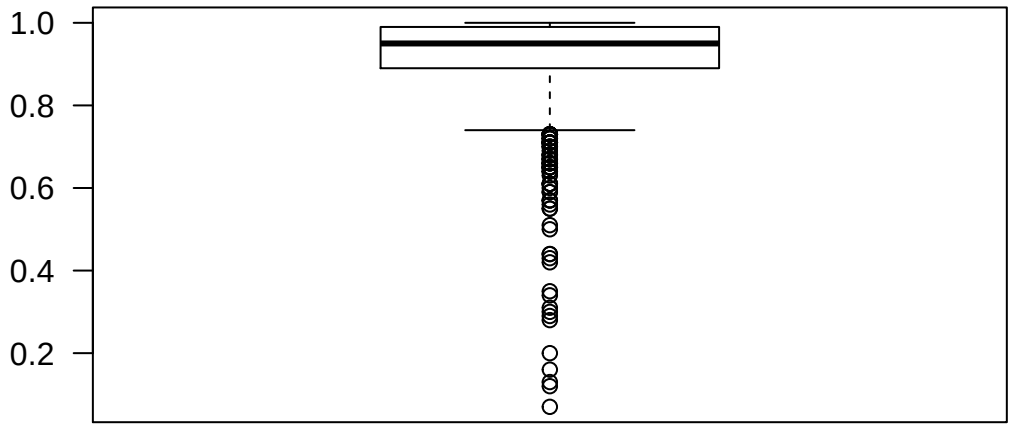

Figure 21: Box plot showing the marginal distribution of the parameter in `paramPhylo`. This plot was constructed using `boxplot`.

References

[1] Otso Ovaskainen et al. "How to make more out of community data? A conceptual framework and its implementation as models and software". In: *Ecology Letters* ().

[2] Jerrold H. Zar. *Biostatistical analysis*. 5th ed. Upper Saddle River, N.J: Prentice-Hall/Pearson, 2010. ISBN: 978-0-13-100846-5.

[3] Tue Tjur. "Coefficients of determination in logistic regression models – A new proposal: The coefficient of discrimination". In: *The American Statistician* 63.4 (2009), pp. 366–372.

[4] Anna Oldén et al. "Bryophyte Species Richness on Retention Aspens Recovers in Time but Community Structure Does Not". en. In: *PLoS ONE* 9.4 (Apr. 2014). Ed. by Keping Ma, e93786. ISSN: 1932-6203. DOI: 10.1371/journal.pone.0093786. URL: http://dx.plos.org/10.1371/journal.pone.0093786 (visited on 02/15/2017).