Hmsc 3.0: Getting started with Hmsc: low-dimensional multivariate models

Gleb Tikhonov Øystein H. Opedal Nerea Abrego Aleksi Lehikoinen
Melinda M. J. de Jonge Jari Oksanen Otso Ovaskainen

19 February 2021

Introduction

The Hierarchical Modelling of Species Communities (HMSC) framework is a statistical framework for analysis of multivariate data, typically from species communities. We assume that the reader has already gone through the vignette "Hmsc 3.0: Getting started with Hmsc: univariate models". Here we continue by demonstrating how to get started with multivariate analyses. We consider here the low-dimensional case, i.e. the case where the number of species is small. The high-dimensional case of a species-rich community is considered in the next vignette "Hmsc 3.0: Getting started with Hmsc: high-dimensional multivariate models".

To get Hmsc-R in use, you need to load it.

```
library(Hmsc)
library(corrplot)
set.seed(1)
```

We also loaded the `corrplot` package which will be used for plotting, and we set the random number seed to make the results presented here reproducible.

Linear model for a community with five species

As the first case study, we use HMSC to fit a multivariate linear model.

Generating simulated data

For illustrative purposes, we use simulated data for which we know the parameter values.

```
n = 100
x1 = rnorm(n)
x2 = rnorm(n)
XData = data.frame(x1=x1,x2=x2)
alpha = c(0,0,0,0,0)
beta1 = c(1,1,-1,-1,0)
beta2 = c(1,-1,1,-1,0)
sigma = c(1,1,1,1,1)
L = matrix(NA,nrow=n,ncol=5)
```

```
Y = matrix(NA,nrow=n,ncol=5)
for (j in 1:5){
   L[,j] = alpha[j] + beta1[j]*x1 + beta2[j]*x2
   Y[,j] = L[,j] + rnorm(n, sd = sigma[j])
}
```

Here, we have generated data for five species, for `n = 100` sampling units (i.e., data points). The environmental predictors `x1` and `x2` are continuous covariates, `alpha`, `beta1` and `beta2` are the true parameters for the intercept and the slopes associated to the two covariates, `L` is the matrix of linear predictors, and `Y` is the matrix of response variables. Indexing the species by j and the sampling units by i, we can write down the model that generated the data as $y_{ij} = \alpha_j + \beta_{1j}x_{1i} + \beta_{2j}x_{2i} + \epsilon_{ij}$, where $\epsilon_{ij} \sim N(0, \sigma_j^2)$. We have assumed that species 1 and 2 respond positively to the covariate x_1 whereas species 3 and 4 respond negatively to it. We have further assumed that species 1 and 3 respond positively to the covariate x_2 whereas species 2 and 4 respond negatively to it. Species five differs from the other species in that it does not respond to either of the covariates. All five species are assumed to have the same amount of residual variation, i.e. variation not explained by the responses to the environmental covariates.

Estimating environmental responses

To analyze these data with HMSC, we construct the model as

```
m = Hmsc(Y = Y, XData = XData, XFormula = ~x1+x2)
```

With the XFormula, we have specified that we assume additive effects of the two covariates x_1 and x_2, i.e. we assume the same model that we used to generate the data.

To fit the HMSC model with Bayesian inference, we use the `sampleMcmc` function. When calling `sampleMcmc`, we need to decide how many chains to sample (`nChains`), how many samples to obtain per chain (`samples`), how long transient (also called burn-in) to include (`transient`), and how frequently we wish to see the progress of the MCMC sampling (`verbose`). MCMC sampling can take a lot of time, so we have included two options below. By setting `test.run = TRUE`, the entire .Rmd version of the vignette can be run quickly, but the parameter estimates will not be reliable. However, that does not matter if the aim of running the vignette is e.g. to get familiar with the syntax of HMSC-R, to examine the structure of the constructed objects and the inputs and outputs of the functions. When setting `test.run = FALSE`, a much larger amount of MCMC sampling is conducted, and running the .Rmd version of the vignette will reproduce the results shown in the .pdf version of the vignette.

```
nChains = 2
test.run = FALSE
if (test.run){
   #with this option, the vignette runs fast but results are not reliable
   thin = 1
   samples = 10
   transient = 5
   verbose = 0
} else {
   #with this option, the vignette evaluates slow but it reproduces the results of the
   #.pdf version
   thin = 10
   samples = 1000
   transient = 500*thin
   verbose = 0
}
```

We are now ready to call `sampleMcmc` and thus estimate the model parameters.

```
m = sampleMcmc(m, thin = thin, samples = samples, transient = transient,
                    nChains = nChains, verbose = verbose)
```

```
## setting updater$GammaEta=FALSE due to absence of random effects included to the model
```

```
## setting updater$latentLoadingOrderSwap=0 disabling full-conditional swapping of consecutive latent l
```

```
## Computing chain 1
## Computing chain 2
```

Note that we set `verbose` to zero to suppress the output reporting how MCMC sampling is proceeding. This was done here just to avoid printing many extra lines to the vignette. Normally one wishes to see the progress, especially if posterior sampling takes a lot of time.

As should be routinely done, we next check MCMC convergence diagnostics.

```
mpost = convertToCodaObject(m)
effectiveSize(mpost$Beta)
```

```
## B[(Intercept) (C1), sp1 (S1)]          B[x1 (C2), sp1 (S1)]
##                       2000.000                       1905.161
##          B[x2 (C3), sp1 (S1)] B[(Intercept) (C1), sp2 (S2)]
##                       2000.000                       2497.156
##          B[x1 (C2), sp2 (S2)]          B[x2 (C3), sp2 (S2)]
##                       2000.000                       2383.868
## B[(Intercept) (C1), sp3 (S3)]          B[x1 (C2), sp3 (S3)]
##                       2000.000                       2236.681
##          B[x2 (C3), sp3 (S3)] B[(Intercept) (C1), sp4 (S4)]
##                       2139.538                       2000.000
##          B[x1 (C2), sp4 (S4)]          B[x2 (C3), sp4 (S4)]
##                       1797.605                       1771.633
## B[(Intercept) (C1), sp5 (S5)]          B[x1 (C2), sp5 (S5)]
##                       2000.000                       2000.000
##          B[x2 (C3), sp5 (S5)]
##                       2500.439
```

```
gelman.diag(mpost$Beta, multivariate=FALSE)$psrf
```

```
##                              Point est. Upper C.I.
## B[(Intercept) (C1), sp1 (S1)]  1.0001206  1.0017339
## B[x1 (C2), sp1 (S1)]           1.0002937  1.0027057
## B[x2 (C3), sp1 (S1)]           1.0001375  1.0033067
## B[(Intercept) (C1), sp2 (S2)]  1.0003117  1.0032300
## B[x1 (C2), sp2 (S2)]           1.0003287  1.0026687
## B[x2 (C3), sp2 (S2)]           1.0045296  1.0218858
## B[(Intercept) (C1), sp3 (S3)]  0.9999867  0.9999982
## B[x1 (C2), sp3 (S3)]           1.0020193  1.0059654
## B[x2 (C3), sp3 (S3)]           1.0014574  1.0076399
## B[(Intercept) (C1), sp4 (S4)]  0.9998040  1.0004751
## B[x1 (C2), sp4 (S4)]           1.0017084  1.0110827
```

```
## B[x2 (C3), sp4 (S4)]           0.9995539  0.9997457
## B[(Intercept) (C1), sp5 (S5)]  1.0021573  1.0117651
## B[x1 (C2), sp5 (S5)]           1.0001996  1.0028535
## B[x2 (C3), sp5 (S5)]           1.0007066  1.0042695
```

As there are five species and three regression parameters per species (including the intercept), there are fifteen parameters for which convergence diagnostics are shown. All looks good: effective sample sizes are high, and potential scale reduction factors are close to one.

As there are quite many parameters, it can be more convenient to look at the convergence diagnostics graphically.

```
par(mfrow=c(1,2))
hist(effectiveSize(mpost$Beta), main="ess(beta)")
hist(gelman.diag(mpost$Beta, multivariate=FALSE)$psrf, main="psrf(beta)")
```

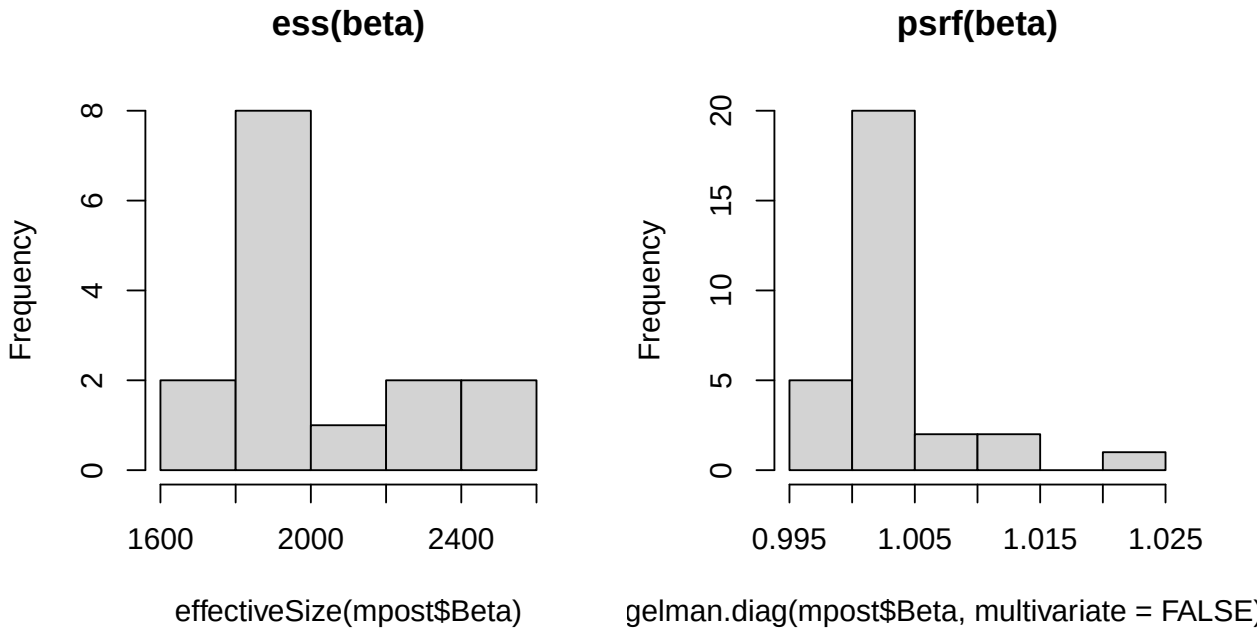

Figure 1: Histograms of effective sample sizes and potential scale reduction factors (psrf) for Beta parameters

We note that thus far we have examined MCMC convergence only for the β-parameters, but the HMSC model has also many other kinds of parameters, and MCMC convergence should be checked for them as well. We will return to this topic below.

To assess the model's explanatory power, we apply the `evaluateModelFit` function to the posterior predictive distribution simulated by the function `computePredictedValues`.

```
preds = computePredictedValues(m)
evaluateModelFit(hM = m, predY = preds)
```

```
## $RMSE
## [1] 1.0278769 0.9840073 1.1407063 0.9557839 1.0672212
##
## $R2
## [1] 0.60901301 0.64199472 0.57468146 0.68038518 0.01542938
```

The model fit is given in terms of R^2 and root-mean-square error `RMSE` for each of the five species. As expected, the explanatory power is poor for the species number five as variation in that species is not related to the environmental covariates. The root-mean-square error is close to one for all species, reflecting the fact that we assumed that the standard deviation of unexplained residual variation is one for each species.

We next evaluate the model's predictive power through two-fold cross validation.

```
partition = createPartition(m, nfolds = 2)
preds = computePredictedValues(m, partition = partition)
```

```
## Cross-validation, fold 1 out of 2

## setting updater$GammaEta=FALSE due to absence of random effects included to the model

## setting updater$latentLoadingOrderSwap=0 disabling full-conditional swapping of consecutive latent l

## Computing chain 1
## Computing chain 2
## Cross-validation, fold 2 out of 2

## setting updater$GammaEta=FALSE due to absence of random effects included to the model
## setting updater$latentLoadingOrderSwap=0 disabling full-conditional swapping of consecutive latent l

## Computing chain 1
## Computing chain 2
```

```
evaluateModelFit(hM = m, predY = preds)
```

```
## $RMSE
## [1] 1.065455 1.042690 1.166040 1.037802 1.117071
##
## $R2
## [1]  0.58087102  0.59936824  0.55678748  0.62371726 -0.01117441
```

The predictive power is not much worse than the explanatory power, as even with cross-validation there is sufficient amount of training data to learn about the environmental responses.

Let us now look at the estimates of the β parameters. We may do so visually by applying the `plotBeta` function.

```
postBeta = getPostEstimate(m, parName = "Beta")
plotBeta(m, post = postBeta, param = "Support", supportLevel = 0.95)
```

In 2, shown by red are those parameters for which the posterior probability for the parameter being positive is greater the chosen support level, here 0.95. Shown in blue are those parameters for which the posterior probability for the parameter being negative is greater than 0.95. The remaining parameters for which there is no strong statistical support for being positive or negative are shown in white. The plot shows that we were able to estimate from the data the parameters that we assumed when generating the data: species 1 and 2 respond positively to the covariate x_1 whereas species 3 and 4 respond negatively to it, species 1 and 3 respond positively to the covariate x_2 whereas species 2 and 4 respond negatively to it, and species 5 does not respond to either of the covariates. As is often the case, the estimate of the intercept is of less interest, as that relates to the mean abundance of each species.

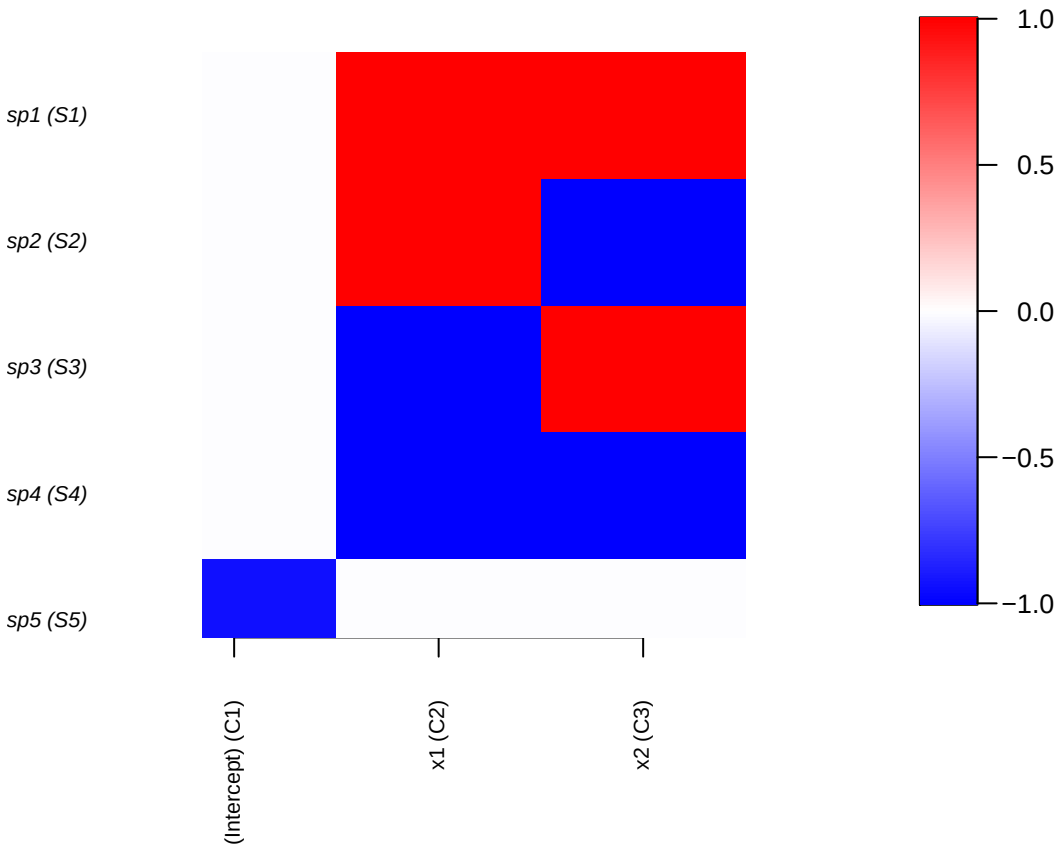

Figure 2: Heatmap of Beta parameters

Estimating species-to-species associations

One of the most important features of HMSC is that it allows estimating species-to-species residual associations, which is implemented through a latent factor approach. We will next extend the above fitted model to include such latent factors. To do so, we need to include a random effect at the level of the sampling unit. We note that in a univariate model, such a random effect would not make sense, as variation related to the random effect would be fully confounded with residual variation. However, in a multivariate model it is possible to define a random effect at the sampling unit level, as in a multivariate case the random effect models not only variation within species, but also co-variation among species, i.e. species-to-species residual associations.

```
studyDesign = data.frame(sample = as.factor(1:n))
rL = HmscRandomLevel(units = studyDesign$sample)
m = Hmsc(Y = Y, XData = XData, XFormula = ~x1+x2,
         studyDesign = studyDesign, ranLevels = list(sample = rL))
m = sampleMcmc(m, thin = thin, samples = samples, transient = transient,
                     nChains = nChains, verbose = verbose)
```

```
## setting updater$latentLoadingOrderSwap=0 disabling full-conditional swapping of consecutive latent l
```

```
## Computing chain 1
## Computing chain 2
```

In a model with random effects, it is important to look at the convergence diagnostics not only for the β parameters, but also for the Ω parameters. The matrix Ω is the matrix of species-to-species residual covariances.

```
mpost = convertToCodaObject(m)
par(mfrow=c(2,2))
hist(effectiveSize(mpost$Beta), main="ess(beta)")
hist(gelman.diag(mpost$Beta, multivariate=FALSE)$psrf, main="psrf(beta)")
hist(effectiveSize(mpost$Omega[[1]]), main="ess(omega)")
hist(gelman.diag(mpost$Omega[[1]], multivariate=FALSE)$psrf, main="psrf(omega)")
```

The convergence diagnostics shown in Fig. 3 are somewhat better for the β parameters than for the Ω parameters. This is typically the case, as it is easier to estimate the fixed effects than the random effects, especially in a multivariate model.

Let us next visualize the estimates of the β parameters.

```
postBeta = getPostEstimate(m, parName="Beta")
plotBeta(m, post=postBeta, param="Support", supportLevel = 0.95)
```

Reassuringly, the estimates shown in Fig. 4 are consistent with those estimated by a model without the random effect.

In addition to the β parameters related to the fixed effects, we can now look at the estimated species-to-species associations. We extract them from the model object with the `computeAssociations` function, which also converts the covariances to the more convenient scale of correlation (ranging from -1 to +1). In the script below, we choose to plot only those associations for which the posterior probability for being negative or positive is at least 0.95. There is no specific function for plotting species-to-species associations in HMSC, but such plots can be generated straightforwardly with the `corrplot` function.

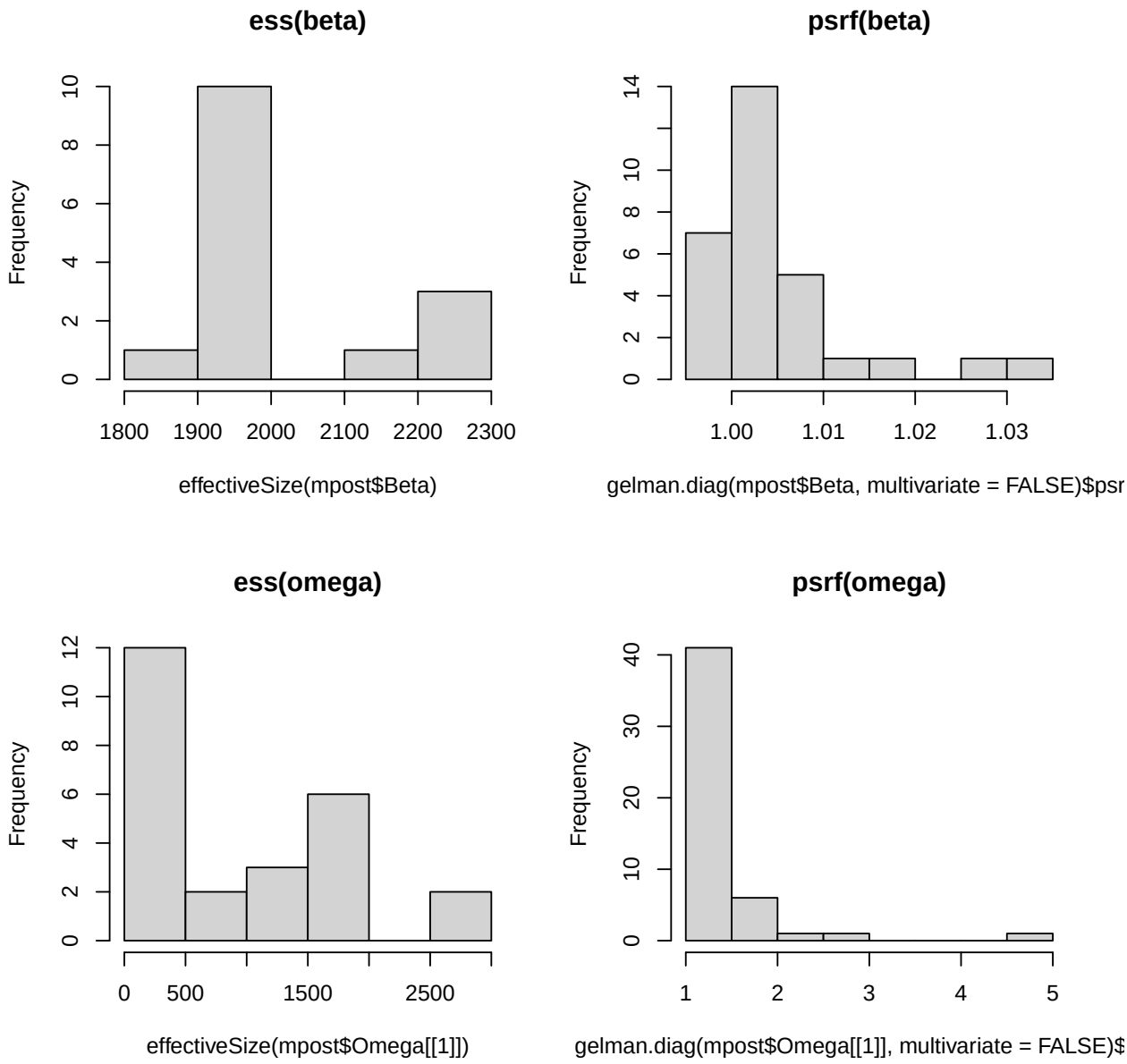

Figure 3: Histograms of effective sample sizes and potential scale reduction factors (psrf) for Beta and Omega parameters

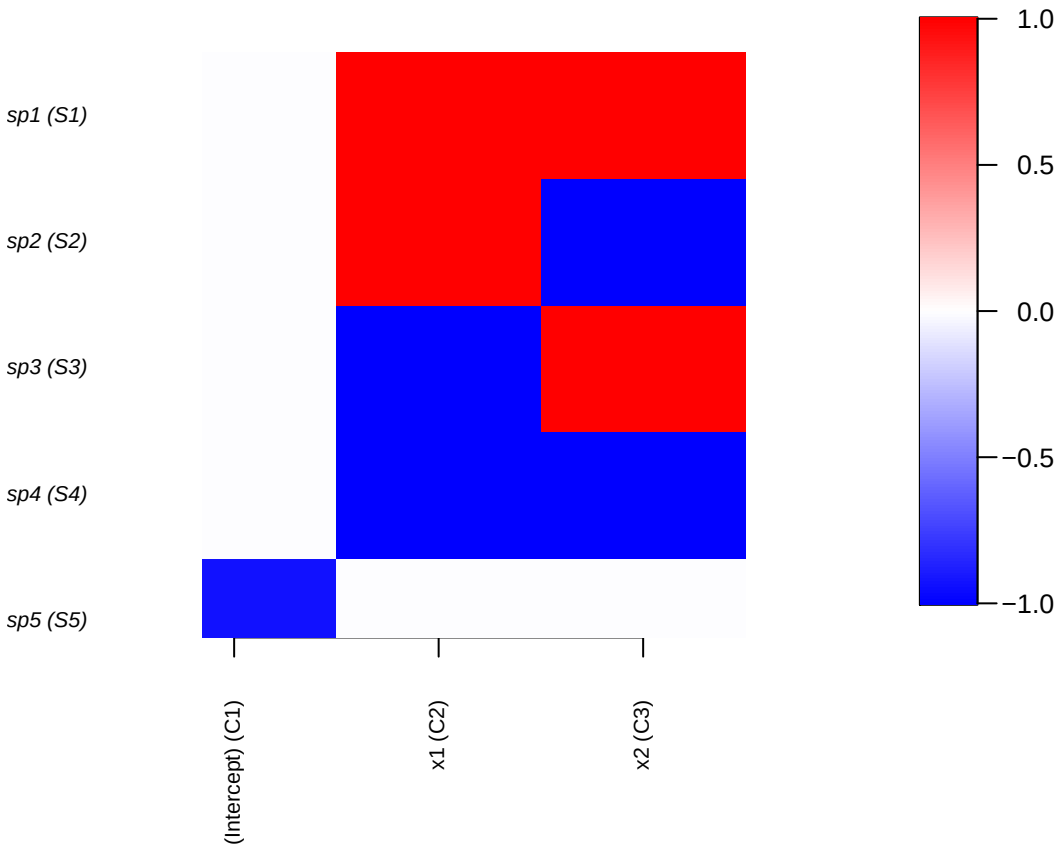

Figure 4: Heatmap of Beta parameters for the multivariate model

```
OmegaCor = computeAssociations(m)
supportLevel = 0.95
toPlot = ((OmegaCor[[1]]$support>supportLevel)
          + (OmegaCor[[1]]$support<(1-supportLevel))>0)*OmegaCor[[1]]$mean
corrplot(toPlot, method = "color",
         col = colorRampPalette(c("blue","white","red"))(200),
         title = paste("random effect level:", m$rLNames[1]), mar=c(0,0,1,0))
```

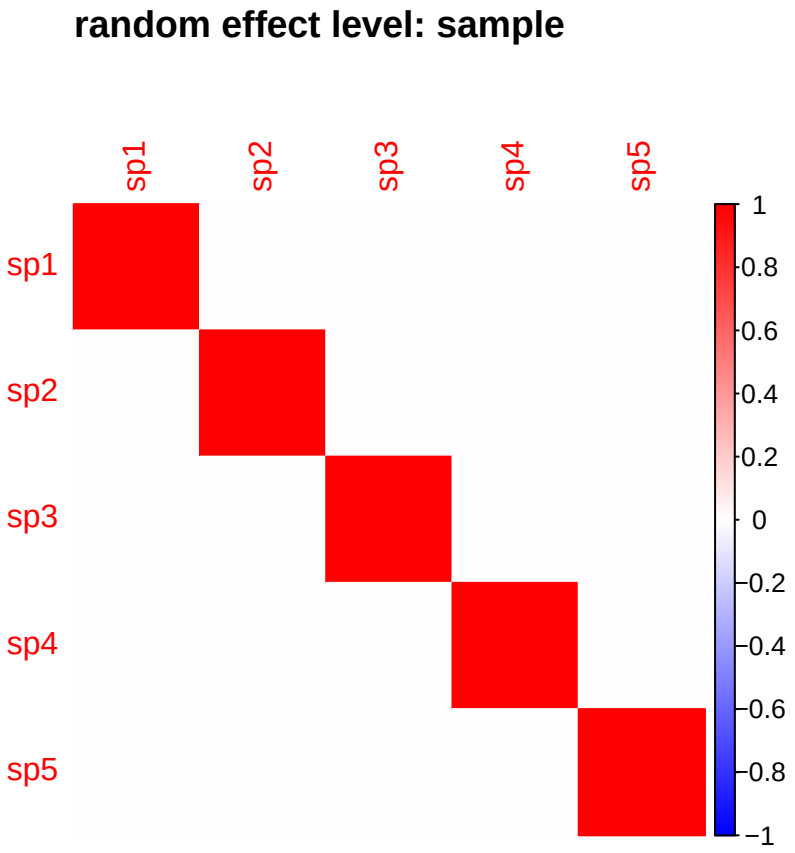

Figure 5: Heatmap of the Omega matrix

The only associations that are visible in Fig. 5 are within-species associations, for which the correlations are always by definition one. This result is fully in line with what we assumed when simulating the data, as we simulated the residuals independently for each species. To make this more transparent, we note that our assumption of $\epsilon_{ij} \sim N(0, \sigma_j^2)$ can be written in the multivariate notation as $\epsilon_{i\cdot} \sim N(0, \Sigma)$, where Σ is a diagonal matrix with the species-specific variances σ_j at the diagonal, and zeros at the off-diagonal.

Let us then repeat the above analyses with a model that is otherwise identical but does not include the covariate x_2. The motivation for dropping x_2 from the model is that often many covariates that do influence the data in reality are not included in the analysis. For example, it might be that the researcher analyzing the data did not think that the covariate x_2 would influence the species abundances, or that it was not possible to measure the covariate x_2 even if it was thought to be relevant.

```
m = Hmsc(Y=Y, XData=XData, XFormula=~x1,
         studyDesign=studyDesign, ranLevels=list(sample=rL))
m = sampleMcmc(m, thin = thin, samples = samples, transient = transient,
                    nChains = nChains, verbose = verbose)
```

```
## setting updater$latentLoadingOrderSwap=0 disabling full-conditional swapping of consecutive latent l

## Computing chain 1
## Computing chain 2
```

```
postBeta = getPostEstimate(m, parName="Beta")
plotBeta(m, post=postBeta, param="Support", supportLevel = 0.95)
```

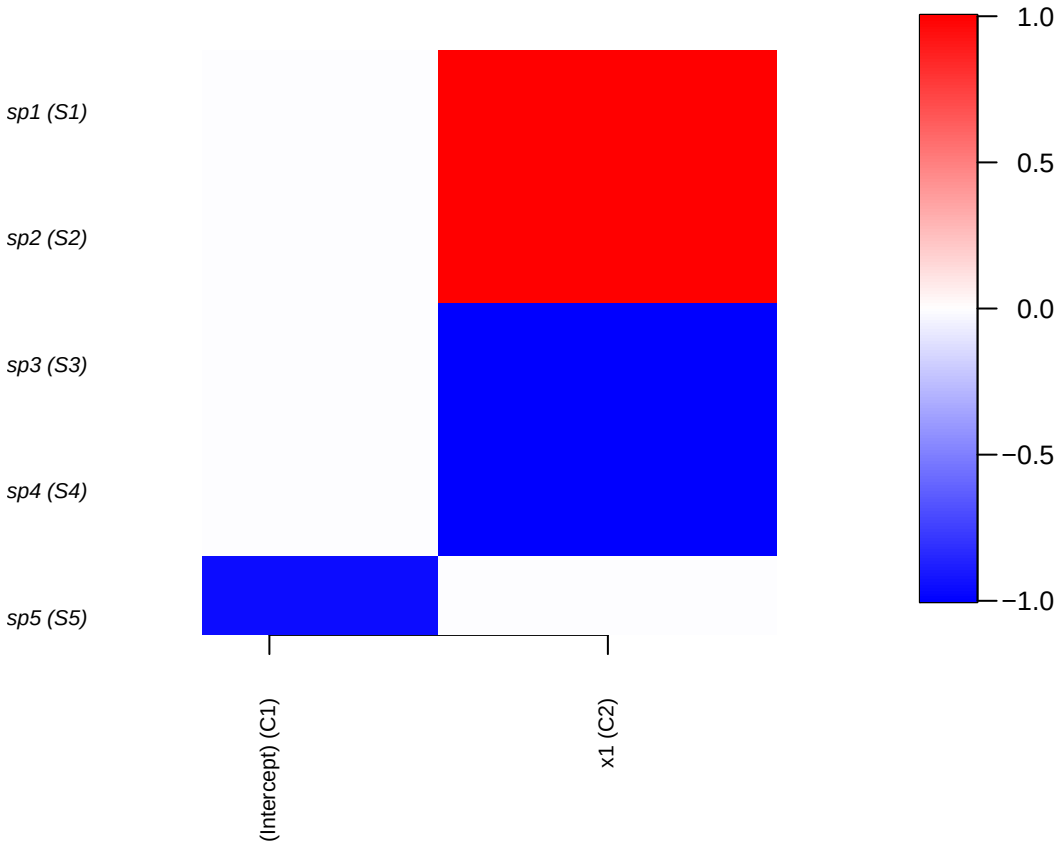

Figure 6: Heatmap of the Beta parameters for the reduced-covariate model

```
OmegaCor = computeAssociations(m)
supportLevel = 0.95
toPlot = ((OmegaCor[[1]]$support>supportLevel)
          + (OmegaCor[[1]]$support<(1-supportLevel))>0)*OmegaCor[[1]]$mean
corrplot(toPlot, method = "color",
         col=colorRampPalette(c("blue","white","red"))(200),
         title=paste("random effect level:", m$rLNames[1]), mar=c(0,0,1,0))
```

The estimates of the β parameters (Fig. 6) show the same responses to the covariate x_1 that was included in the reduced model as the full model that contained both covariates. But what is now different is that the association matrix (Fig. 7) shows residual correlations among the species: species 1 and 3 are positively correlated with each other, and so are species 2 and 4, but these two groups of species are negatively correlated with each other. In contrast, species 5 is not correlated with any other species. The reason for these residual correlations is the responses of the species to the covariate x_2, which is missing from the model, and thus its influence is seen in the residual associations. Species 1 and 3 were assumed to respond similarly (both

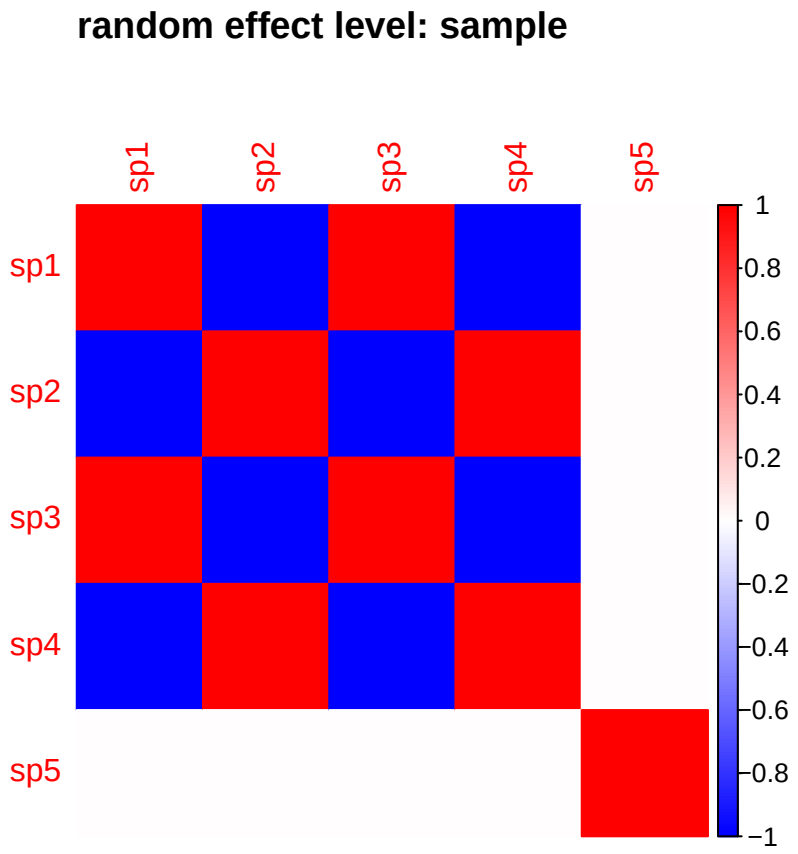

Figure 7: Heatmap of the Omega matrix from the reduced-covariate model

positively) to the missing covariate x_2, creating a positive association among them. Species 1 and 2 responded dissimilarly (one positively and one negatively) to the missing covariate x_2, creating a negative association among them.

In general, residual associations among species can be generated by correlated responses to missing covariates, or by ecological interactions. While HMSC analyses can be used to identify residual associations among the species, they cannot be used to separate conclusively whether the associations are due to missing covariates or due to ecological interactions. This is not a limitation of HMSC *per se*, but a limitation of the type of data that is used as input: identical data on species occurrences or abundances can be generated either by responses to missing covariates or by ecological interactions. To ascertain whether the associations are due to missing covariates or ecological interactions, some other type of data would thus be needed. As one example, one could conduct experiments where the fitnesses of the species are measured when they are or are not allowed to interact with each other. In the absence of such additional data, the best that can be done is to control in the model for those environmental covariates that can be expected (or are shown by cross-validation) to be relevant, and then interpret the remaining residual associations with caution, and keeping the ecological knowledge about the study organisms in mind.

Explanatory power, predictive power, and conditional predictive power

The model where the responses to a missing covariate are captured by residual associations provides us an interesting case study to discuss different ways in which explanatory and predictive power can be computed. Let us start by evaluating the explanatory power.

```
preds = computePredictedValues(m)
evaluateModelFit(hM = m, predY = preds)
```

```
## $RMSE
## [1] 0.9700327 1.0429795 0.8772745 0.8248486 1.0669777
##
## $R2
## [1] 0.66621524 0.61244054 0.77944310 0.77675422 0.01804719
```

The explanatory power of the reduced model (that has only x_1 as the covariate) is similar to the explanatory power of the full model (that has both x_1 and x_2 as covariates). This may appear surprising, as we generated the data with the assumption that the species respond equally strongly to the missing covariate x_2 as to the covariate x_1 included in the model. The equally high explanatory power is due to the fact that the random effect part of the model also helps to make the predictions. The underlying latent variable approach can actually be viewed as a way to *estimate* missing covariates (latent variables) and species' responses to them (latent loadings). Thus, the fitted model has information about the values of the missing covariates in the sampling units used for model fitting, and this information can be used when making the predictions.

We next evaluate the predictive power of the model by cross-validation.

```
preds = computePredictedValues(m, partition = partition)
```

```
## Cross-validation, fold 1 out of 2
```

```
## setting updater$latentLoadingOrderSwap=0 disabling full-conditional swapping of consecutive latent l
```

```
## Computing chain 1
## Computing chain 2
## Cross-validation, fold 2 out of 2
```

```
## setting updater$latentLoadingOrderSwap=0 disabling full-conditional swapping of consecutive latent lo

## Computing chain 1
## Computing chain 2
```

```
evaluateModelFit(hM = m, predY = preds)
```

```
## $RMSE
## [1] 1.370556 1.451486 1.599429 1.374369 1.118346
##
## $R2
## [1]  0.30465539  0.22213690  0.16342324  0.33851668 -0.02711607
```

The predictive power is much lower than the explanatory power. This is because the fitted model cannot know the values of the missing covariates (technically, the latent variables) for the sampling units not included in model fitting, and thus the predictive power of the model to new sampling units is based on the covariate x_1 only.

With the multivariate model, it is also possible to generate conditional predictions. When generating conditional predictions for a focal species, the observed data for some other species are assumed to be known. If the focal species and the other species show residual associations, knowing the observed data for the other species can help to make improved predictions for the focal species. The possibility of making conditional predictions in the context of cross-validation has been implemented in the `computePredictedValues` function. In addition to the `partition` among the sampling units, we now apply also `partition.sp` over the species.

```
preds = computePredictedValues(m, partition=partition,
                               partition.sp=c(1,2,3,4,5), mcmcStep=10)
```

```
## Cross-validation, fold 1 out of 2

## setting updater$latentLoadingOrderSwap=0 disabling full-conditional swapping of consecutive latent lo

## Computing chain 1
## Computing chain 2
## Cross-validation, fold 2 out of 2

## setting updater$latentLoadingOrderSwap=0 disabling full-conditional swapping of consecutive latent lo

## Computing chain 1
## Computing chain 2
```

```
evaluateModelFit(hM=m, predY=preds)
```

```
## $RMSE
## [1] 1.170191 1.359334 1.377879 1.228176 1.120498
##
## $R2
## [1]  0.49618432  0.31816252  0.38236941  0.47441626 -0.03025794
```

By writing `partition.sp=c(1,2,3,4,5)` we chose to do a full leave-one-out cross validation across the five species, in combination with the two-fold cross validation (defined by `partition`) over the sampling units. With these choices, the predicted values are computed as follows.

First, the fold 1 of sampling units is considered as the focal fold for which predictions are to be generated. Thus, the model is fitted using only sampling units from fold 2 as training data. The predictions are then made separately for each focal species. For example, the predictions for species 1 are made conditional on the observed data for the other species 2-5. At this stage, the observed data on the species 2-5 are utilized to estimate the latent variables for the focal sampling unit. The focal species 1 is then assumed to respond to the latent variables based on its latent loadings. As shown by the improved measures of model fit, the predictions based on conditional cross-validation are more accurate than the predictions based on the 'normal' cross-validation.

Making conditional predictions (and thus conditional cross-validations) requires the estimation of the latent variables for the sampling units for which the predictions are to be generated. This can in practice be done with the help of MCMC sampling, which is why the parameter `mcmcStep` is included in the function call. We have here selected to do the conditional predictions by updating the latent variables with 10 MCMC iterations. There is no general rule on how many MCMC iterations will be sufficient, and thus we recommend setting `mcmcStep` first e.g. to 10 and then to 100 to see if the results improve. As always, running more MCMC iterations means a longer computational time.

We note that conditional prediction is conceptually similar to using non-focal species as predictors. For example, the conditional predictions by the multivariate model for species 1 are very similar to what would be predicted by the univariate model `y1~x1+y2+y3+y4+y5`. While including the other species as predictors is a viable option for communities with small number of species, it is not straightforward to do if there are tens or hundreds of species. In contrast, as we will show in the vignette "HMSC-R 3.0: Getting started with HMSC-R: high-dimensional multivariate models", conditional predictions can be made also for large species communities.

Model-based ordination

Another way to visualize the responses of the species to the latent variables is to construct a biplot, as commonly done in ordination analyses. First, we fit the model without any covariates, and the number of latent factors constrained to 2.

```
rL$nfMin=2
rL$nfMax=2

m = Hmsc(Y=Y, XData=XData, XFormula=~1,
         studyDesign=studyDesign, ranLevels=list(sample=rL))
m = sampleMcmc(m, thin = thin, samples = samples, transient = transient,
                     nChains = nChains, verbose = verbose)
```

```
## setting updater$latentLoadingOrderSwap=0 disabling full-conditional swapping of consecutive latent l
```

```
## Computing chain 1
## Computing chain 2
```

We now extract the posterior means of the η and λ parameters, which represent the site loadings (η) and species loadings (λ) on the latent factors. We then assign colors to the site loadings according to the value of the covariate x_2 used in simulating the data (but that was not used in model fitting).

```
etaPost=getPostEstimate(m, "Eta")
lambdaPost=getPostEstimate(m, "Lambda")

biPlot(m, etaPost = etaPost, lambdaPost = lambdaPost, factors = c(1,2), "x2")
```

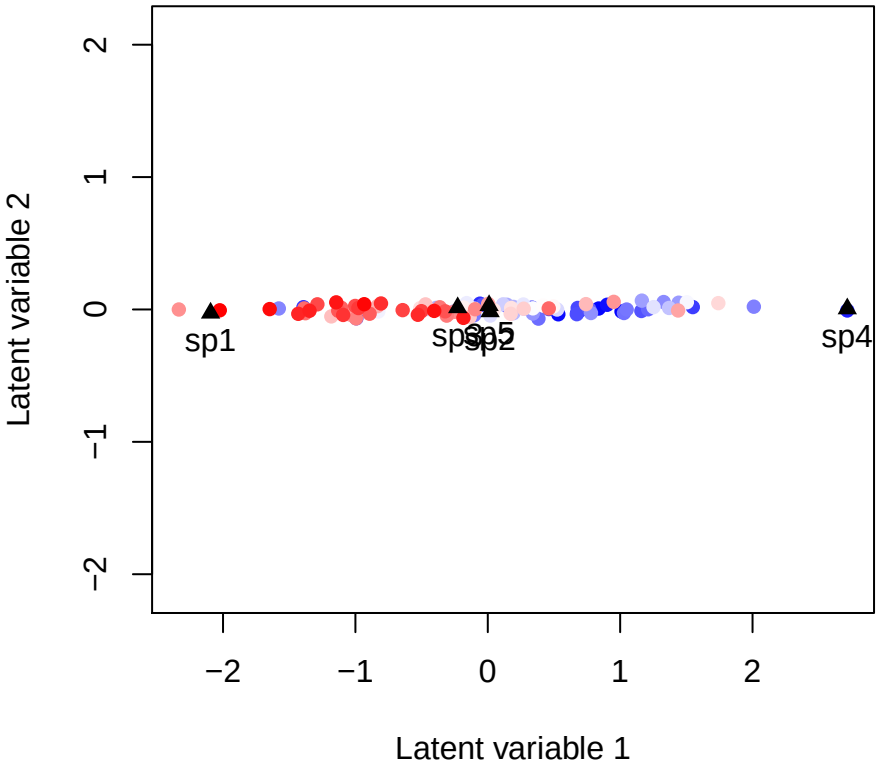

Figure 8: An ordination biplot of the site loadings (circles) and the species loadings (triangles).

In the biplot of Fig. 8, the sites are clearly ordered by the color, which reflects the fact the latent variable has captured the missing covariate x_2. Species 1 and 3 are close to each other in this plot, as are species 2 and 4, reflecting their similar responses to the missing covariate.

A community of four species with a mixture of data types

As the second case study, we use HMSC to fit a multivariate non-linear model. While all species are typically assumed to follow the same error distribution, HMSC allows also for mixtures of them. To illustrate, we assume the same environmental covariates and the same linear predictors as we did for species 1-4 in the example above, but we apply four different link functions and error distributions. More precisely, we will assume that species 1 follows the normal model, species 2 the probit model, species 3 the Poisson model, and species 4 the log-normal Poisson model. Perhaps we have measured the abundance of species 1 as biomass and the abundances of species 3 and 4 as counts of individuals, whereas for species 2 only presence-absence data are available.

```
set.seed(2)
alpha = c(0,0,0,0)
beta1 = c(1,1,-1,-1)
beta2 = c(1,-1,1,-1)
```

```
sigma = c(1,NA,NA,1)
L = matrix(NA,nrow=n,ncol=4)
Y = matrix(NA,nrow=n,ncol=4)
for (j in 1:4){
   L[,j] = alpha[j] + beta1[j]*x1 + beta2[j]*x2
}
Y[,1] = L[,1] + rnorm(n, sd = sigma[1])
Y[,2] = 1*((L[,2] + rnorm(n, sd = 1))>0)
Y[,3] = rpois(n, lambda = exp(L[,3]))
Y[,4] = rpois(n, lambda = exp(L[,4] + rnorm(n, sd = sigma[4])))
```

Note that we have not defined the residual variance `sigma` for species 2 and 3 because this parameter is irrelevant for the probit and Poisson models. Let us illustrate the nature of the data by viewing it for the first 10 sampling units.

```
Y[1:10,]
```

```
##              [,1] [,2] [,3] [,4]
##  [1,] -2.1437350    1    1    3
##  [2,]  0.4106084    1    1    1
##  [3,] -0.1587049    0    0    3
##  [4,]  0.6229339    1    0    0
##  [5,] -0.4053286    1    3    5
##  [6,]  1.0792392    0   16    0
##  [7,]  1.9120913    1    1    2
##  [8,]  1.4088009    1    0    0
##  [9,]  2.9444406    0    2    1
## [10,]  1.2380007    0    8    0
```

While we could repeat for these data all the analyses that we did for the linear model of five species, to avoid repetition we perform here only some basic analyses with a HMSC model without random effect. To account for the different types of response variables, we define it as follows.

```
m = Hmsc(Y = Y, XData = XData, XFormula = ~x1+x2,
         distr = c("normal","probit","poisson","lognormal poisson"))
```

If all response variables would have followed the same error distribution, it would have been sufficient to set the error distribution only once, e.g. as `distr="probit"`. Now that we have assumed different types of response variables, we define `distr` as a vector where each element corresponds to each species.

We next fit the model and check the MCMC diagnostics.

```
m = sampleMcmc(m, thin = thin, samples = samples, transient = transient,
                    nChains = nChains, verbose = verbose)
```

```
## setting updater$GammaEta=FALSE due to absence of random effects included to the model

## setting updater$latentLoadingOrderSwap=0 disabling full-conditional swapping of consecutive latent l

## Computing chain 1
## Computing chain 2
```

```
mpost = convertToCodaObject(m)
effectiveSize(mpost$Beta)
```

```
## B[(Intercept) (C1), sp1 (S1)]          B[x1 (C2), sp1 (S1)]
##                     2000.00000                    1936.97107
##          B[x2 (C3), sp1 (S1)] B[(Intercept) (C1), sp2 (S2)]
##                     2000.00000                    2000.00000
##          B[x1 (C2), sp2 (S2)]          B[x2 (C3), sp2 (S2)]
##                     1655.23197                    1634.11522
## B[(Intercept) (C1), sp3 (S3)]          B[x1 (C2), sp3 (S3)]
##                       43.90120                      92.94714
##          B[x2 (C3), sp3 (S3)] B[(Intercept) (C1), sp4 (S4)]
##                       98.29761                     155.71894
##          B[x1 (C2), sp4 (S4)]          B[x2 (C3), sp4 (S4)]
##                      331.92072                     173.41082
```

```
gelman.diag(mpost$Beta, multivariate=FALSE)$psrf
```

```
##                               Point est. Upper C.I.
## B[(Intercept) (C1), sp1 (S1)]  0.9995047  1.0001389
## B[x1 (C2), sp1 (S1)]           0.9993445  0.9993606
## B[x2 (C3), sp1 (S1)]           0.9996190  0.9996368
## B[(Intercept) (C1), sp2 (S2)]  1.0021172  1.0043175
## B[x1 (C2), sp2 (S2)]           0.9996858  1.0011004
## B[x2 (C3), sp2 (S2)]           0.9996481  1.0009110
## B[(Intercept) (C1), sp3 (S3)]  1.0878406  1.2609524
## B[x1 (C2), sp3 (S3)]           1.0153527  1.0469184
## B[x2 (C3), sp3 (S3)]           1.0305760  1.0325761
## B[(Intercept) (C1), sp4 (S4)]  1.0751060  1.2955360
## B[x1 (C2), sp4 (S4)]           1.0199400  1.0936827
## B[x2 (C3), sp4 (S4)]           1.0150389  1.0588013
```

As expected from the corresponding results from the univariate analyses, the MCMC convergence is best for the normally distributed response variable and worst for the Poisson distributed response variable. In a more serious analysis, we should increase the thinning to obtain better MCMC convergence.

As always, we can measure the model's explanatory power with the function `evaluateModelFit`.

```
preds = computePredictedValues(m, expected = FALSE)
evaluateModelFit(hM = m, predY = preds)
```

```
## $RMSE
## [1] 1.1403918 0.3764137 1.4247807 5.2385590
##
## $R2
## [1] 0.5146917        NA        NA        NA
##
## $AUC
## [1]        NA 0.8841537        NA        NA
##
## $TjurR2
## [1]        NA 0.4136086        NA        NA
```

```
##
## $SR2
## [1]        NA        NA 0.6120680 0.5814539
##
## $O.AUC
## [1]        NA        NA 0.8791233 0.8550347
##
## $O.TjurR2
## [1]        NA        NA 0.4019635 0.3389661
##
## $O.RMSE
## [1]        NA        NA 0.3702789 0.3850142
##
## $C.SR2
## [1]        NA        NA 0.5542849 0.4949378
##
## $C.RMSE
## [1]       NA       NA 1.656983 6.410790
```

As discussed in the vignette on univariate models, the types of performance measures that can be computed depend on the nature of the response data. For this reason, many of the performance measures can not be evaluated for all species, and thus their values are `NA`, i.e. missing.

Even if the species follow different error distributions, they all have the underlying linear predictor at the same scale, and the same types of underlying parameters. Thus, we may for example view the β parameters (Fig. 9) for all species simultaneously, as we did for the case where all species followed a normal model.

```
postBeta = getPostEstimate(m, parName="Beta")
plotBeta(m, post=postBeta, param="Support", supportLevel = 0.95)
```

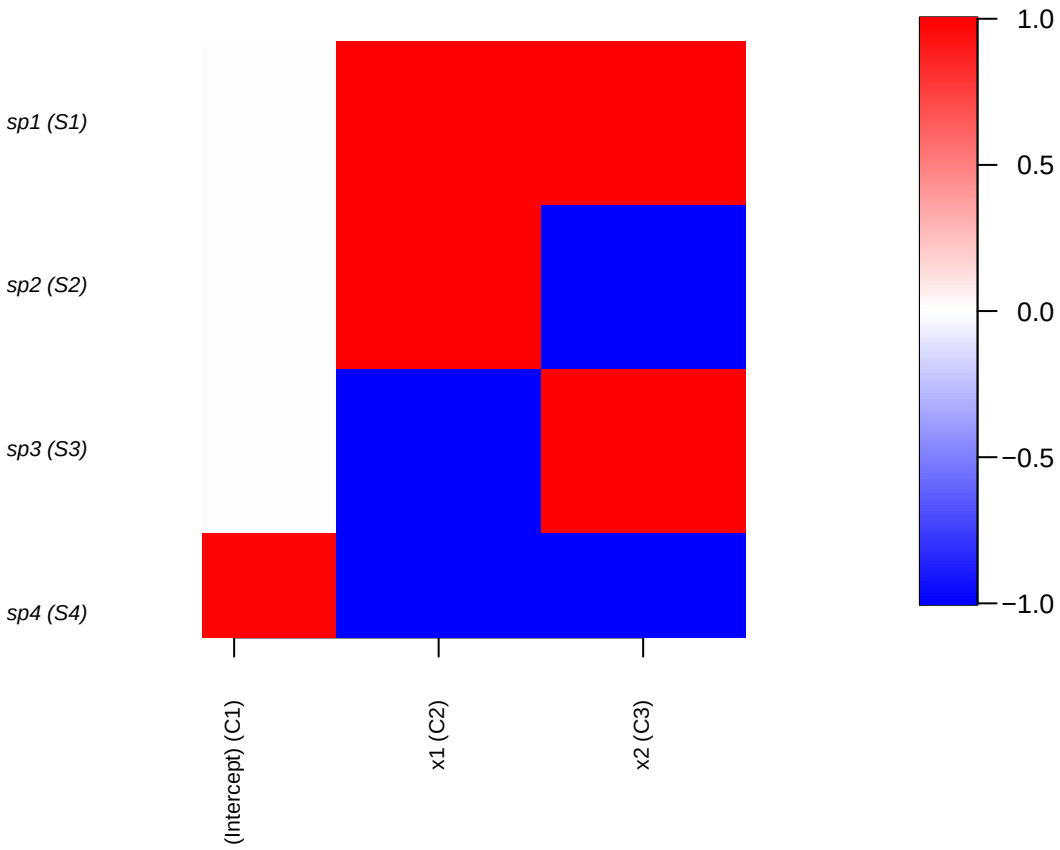

Figure 9: Heatmap of the Beta parameters for the mixed-distributions model