HMSC-R 3.0:
Hierarchical Modelling of Species Communities

Gleb Tikhonov, Øystein H. Opedal, Nerea Abrego, Aleksi Lehikoinen & Otso Ovaskainen

11 March 2019

Introduction

Hierarchical Modelling of Species Communities (HMSC) is a statistical framework for analysis of multivariate data, typically from species communities. It uses Bayesian inference to fit latent-variable joint species distribution models. The conceptual basis of the method is outlined in (Ovaskainen et al. 2017). Here, we demonstrate basic use of the `HMSC-R 3.0` package.

Running a typical HMSC analysis with `HMSC-R` includes five main steps: (1) Setting model structure and fitting the model, (2) Examining MCMC convergence, (3) Evaluating model fit, (4) Exploring parameter estimates, and (5) Making predictions. To demonstrate the workflow, we will analyse a dataset of bird communities in Finland.

(1) Setting model structure and fitting the model

Set directories and load packages

```r
setwd("C:/data/HMSC-R/vignette_birds")
localDir = "."
dataDir = file.path(localDir, "data")
ModelDir = file.path(localDir, "models")
MixingDir = file.path(localDir, "mixing")
MFDir = file.path(localDir, "model_fit")
source("load_libraries.r")
library(knitr)
```

Read data matrices

The minimum data requirement for running a HMSC analysis is presence or abundance data of at least one species (**Y** matrix), and a matrix of environmental covariates (**X** matrix). If presence or abundance data are available for at least two species, the model can be fitted with no environmental covariates and then corresponds to a model-based ordination analysis.

In addition, the HMSC framework allows including trait and phylogenetic data. These data are used to inform the estimation of species' responses to environmental covariates. The effects of traits (**T** matrix) are modelled as linear regression slopes of environmental responses of each species on their trait values. The effect of phylogeny (**C** matrix) is modelled assuming that species' environmental responses follow a multivariate normal distribution with a phylogenetic signal in the variance matrix. The phylogeny can be supplied directly as a phylogenetic distance matrix, or as a phylogenetic tree.

To implement spatially explicit latent factors (Ovaskainen et al. 2016), we also need a matrix of geographical coordinates for each site (**xy**).

```r
# SPECIES AND ENVIRONMENTAL DATA
data = read.csv(file.path(dataDir, "data.csv"))

# SPECIES DATA
Y = as.matrix(data[,10:59])

# ENVIRONMENTAL COVARIATES
XData = data[,c(5,6,7,8,9)]

# PHYLOGENY
phyloTree <- ape::read.tree(file.path(localDir,"data", "CTree.tre"))

# TRAITS
TrData = read.csv(file.path(dataDir, "traits.csv"))
TrData$LogMass = log(TrData$Mass)
```

Set up the model

HMSC handles an arbitrary number of random effects, that be spatially explicit, hierarchical, or both. In the current example there is one spatial random effect (Route). The dataframe `studyDesign` describes the assignment of sampling units (rows in $\mathbf{Y}$) to levels of each random effect. Further structure of each random effect (e.g. spatial coordinates) are assigned using `HmscRandomLevel`. Here, we modify the default prior using `setPriors` by constraining the minimum and maximum number of latent factors.

```r
# STUDY DESIGN
studyDesign = matrix(NA,nrow(Y),2)
studyDesign[,1] = sprintf('Route_%.3d',data$Route)
studyDesign[,2] = sprintf('Year_%.3d',data$Year)
studyDesign = as.data.frame(studyDesign)
colnames(studyDesign) = c("Route","Year")
studyDesign[,1]=as.factor(studyDesign[,1])
studyDesign[,2]=as.factor(studyDesign[,2])

# RANDOM EFFECT STRUCTURE, HERE ROUTE AS A SPATIAL LATENT VARIABLE
routes = levels(studyDesign[,1])
nroutes = length(routes)
xy = matrix(0, nrow = nroutes, ncol = 2)
for (i in 1:nroutes){
   rows=studyDesign[,1]==routes[[i]]
   xy[i,1] = mean(data[rows,]$x)
   xy[i,2] = mean(data[rows,]$y)
}
colnames(xy) = c("x","y")
sRL = xy
rownames(sRL) = routes
rL = HmscRandomLevel(sData=sRL)
rL = setPriors(rL, nfMin = 5, nfMax = 10)
```

Covariates can be supplied as a user-formatted matrix `X`. Alternatively, a dataframe `XData` can be supplied containing the variables to be included in the model, and possibly other variables not included in the model. In the latter case, variables to be included are specified using familiar R formula syntax. Both `X` and `Xdata` may contain continuous variables, categorical variables, and interactions. Here, `AprMay` is a continuous covariate fitted with a linear term and a second-order polynomial term, and `Habitat` is a categorical variable.

```
XFormula = ~ Habitat + poly(AprMay,degree = 2,raw = TRUE)
```

Similarly, trait data can be supplied in the dataframe `TrData`, with the traits to be included in the model specified using `TrFormula`.

```
TrFormula = ~Migration + LogMass
```

To construct the model, we use the `Hmsc` function. We demonstrate model setup for the abundance model with environmental and spatial predictors. The argument 'distr' sets the error distribution of the analysis, here lognormal Poisson. Binomial (distr = "probit"") and Gaussian (distr ="normal") errors are also available.

```
    m = Hmsc(Y = Y,
             XData = XData,  XFormula = XFormula,
             TrData = TrData, TrFormula = TrFormula,
             phyloTree = phyloTree,
             distr = "lognormal poisson",
             studyDesign = studyDesign, ranLevels = list(Route=rL))
```

Run MCMC and save the model

The MCMC sampling scheme can be controlled by adjusting the desired number of posterior samples, the thinning interval (number of steps of the MCMC chain between each sample), the length of the transient to be cut from the chain before sampling, and the number of chains. `adaptNf` controls the number of iterations during which the number of latent factors are adapted. Depending on the length of the chain and the complexity of the model, it may take a long time to run. It is therefore always recommended to save the outputs to a local file. A good strategy can be first to run the model with `thin = 1` to obtain inital results reasonably fast, and then increase the thinning interval (e.g. to 10 and then 100) until model convergence is adequate and results stabilise.

```
thin = 100
samples = 1000
nChains = 4
    set.seed(1)
    ptm = proc.time()
        m = sampleMcmc(m, samples = samples, thin = thin,
                       adaptNf = rep(ceiling(0.4*samples*thin),1),
                       transient = ceiling(0.5*samples*thin),
                       nChains = nChains, nParallel = nChains,
                       initPar = "fixed effects")
    computational.time =  proc.time() - ptm

    filename = file.path(ModelDir, paste("model_",as.character(model),"_",
               c("pa","abundance")[modeltype],"_thin_", ... = as.character(thin),
               "_samples_", as.character(samples),".Rdata",sep = ""))
    save(m,file=filename,computational.time)
```

(2) Examining MCMC convergence

A key step in MCMC-based Bayesian analysis is to assess the performance of the sampling procedure by evaluating chain mixing and convergence. This can be easily done in R using tools from the `coda` package. Within `HMSC-R`, the function `convertToCodaObject` converts the posterior distributions produced by `HMSC-R` into `coda`-format.

Compute mixing statistics

Chain mixing can be evaluated, for example, by assessing the effective size of the posterior sample, i.e. the sample size controlled for autocorrelation among sequential posterior samples (Figure 1). The effective sample size can be extracted using the `coda::effectiveSize` function. Here, we use `for`-loops to sequentially load and process each of the six models.

```
thin = 100
samples = 1000
nChains = 4
comp.time = matrix(nrow=2, ncol=3)
for (modeltype in 1:2){
  for (model in 1:3){
    filename = file.path(ModelDir, paste("model_",as.character(model),"_",
                                         c("pa","abundance")[modeltype],
                                           "_chains_",as.character(nChains),
                                         "_thin_", as.character(thin),"_samples_",
                                         as.character(samples),
                                         ".Rdata",sep = ""))
    load(filename)
    comp.time[modeltype,model] = computational.time[1]

    mpost = convertToCodaObject(m)

    es.beta = effectiveSize(mpost$Beta)
    ge.beta = gelman.diag(mpost$Beta,multivariate=FALSE)$psrf
    es.gamma = effectiveSize(mpost$Gamma)
    ge.gamma = gelman.diag(mpost$Gamma,multivariate=FALSE)$psrf
    es.rho = effectiveSize(mpost$Rho)
    ge.rho = gelman.diag(mpost$Rho,multivariate=FALSE)$psrf
    es.V = effectiveSize(mpost$V)
    ge.V = gelman.diag(mpost$V,multivariate=FALSE)$psrf

    if (model==2){
      es.omega = NA
      ge.omega = NA
    } else {
      es.omega = effectiveSize(mpost$Omega[[1]])
      ge.omega = gelman.diag(mpost$Omega[[1]],multivariate=FALSE)$psrf
    }

    mixing = list(es.beta=es.beta, ge.beta=ge.beta,
                  es.gamma=es.gamma, ge.gamma=ge.gamma,
                  es.rho=es.rho, ge.rho=ge.rho,
                  es.V=es.V, ge.V=ge.V,
                  es.omega=es.omega, ge.omega=ge.omega)
    filename = file.path(MixingDir, paste("mixing_",as.character(model),"_",
                                          c("pa","abundance")[modeltype],
                                          "_chains_",as.character(nChains),
                                          "_thin_", as.character(thin),"_samples_",
                                          as.character(samples),
                                          ".Rdata",sep = ""))
    save(file=filename, mixing)
  }}
```

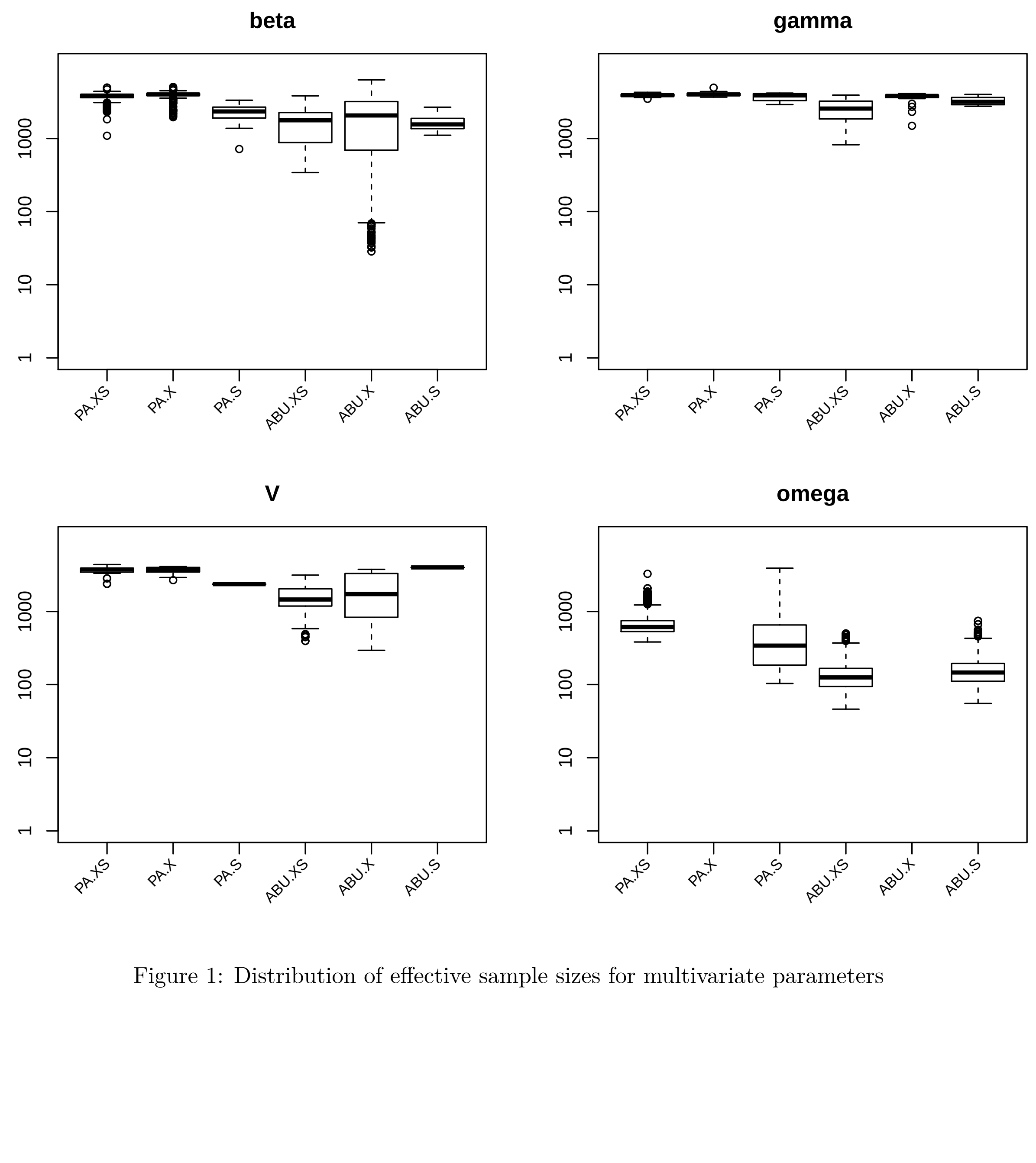

Figure 1: Distribution of effective sample sizes for multivariate parameters

(3) Evaluating model fit

Explanatory power

The explanatory power of the model (Figure 2, grey bars) can be evaluated by different measures of model fit, including RMSE (root mean square error), R^2 (coefficient of determination), Tjur R^2 (coefficient of discrimination), and AUC (area under the receiver operating characteristic curve). In `HMSC-R`, these are returned by the `evaluateModelFit` function.

- `computePredictedValues` returns model-predicted values for the original sampling units used in model fitting.
- `evaluateModelFit` returns several measures of model fit, including RMSE (root mean square error), R^2 (coefficient of determination), Tjur R^2 (coefficient of discrimination), and AUC (area under the receiver operating characteristic curve).

```r
thin = 100
samples = 1000
nChains = 4
for(modeltype in c(1,2)){
  for (model in c(1,2,3)){
    filename = file.path(ModelDir, paste("model_",as.character(model),
                                          c("_pa","_abundance")[modeltype],
                                           "_chains_",as.character(nChains),
                                          "_thin_", as.character(thin),
                                          "_samples_", as.character(samples),
                                          ".Rdata",sep = ""))
    load(filename)
    set.seed(1)
    predY = computePredictedValues(m, expected=FALSE)
    MF = evaluateModelFit(hM=m, predY=predY)
    filename = file.path(MFDir, paste("model_",as.character(model),
                                          c("_pa","_abundance")[modeltype],
                                          "_chains_",as.character(nChains),
                                          "_thin_", as.character(thin),
                                          "_samples_", as.character(samples),
                                          ".Rdata",sep = ""))
    save(file=filename, MF)
  }
}
```

Predictive power

The predictive power of the model (Figure 2, red bars) can be evaluated by cross-validation, where the model is refitted to a subset of data and predictions made for sites not included in the model fit. The same measures of model fit can be computed as for explanatory power. We use the `createPartition` function to split the data into subsets (‘folds’, here 4) used sequentially in model fitting, and pass this as an argument to `computePredictedValues`. The `column` argument controls which random factor (here ‘Route’) is sampled for the partition. For each random level included in a given parition, all data (sampling units) are included.

For smaller data sets, it is feasible to perform leave-one-out cross-validation (nfolds = number of sampling units), with predictions for each site made from a model excluding only the focal site. However, because cross-validation involves refitting the model for each fold, this may not be computationally feasible for larger data sets. On the other extreme, two-fold cross-validation (nfolds=2, half of the original data used for model fitting) takes approximately as long as fitting the original model, but may be overly pessimistic in terms

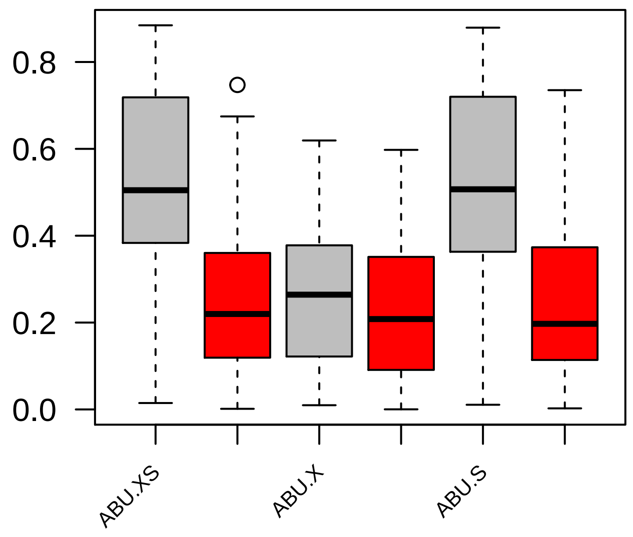

Figure 2: Explanatory (grey bars) and predictive (red bars) power as evaluated by R^2

of predictive power. One strategy is therefore to first perform two-fold cross-validation, then increase the number of folds and assess how results change.

```
for(modeltype in c(1,2)){
  for (model in c(1,2,3)){
    filename = file.path(ModelDir, paste("model_",as.character(model),
                                         c("_pa","_abundance")[modeltype],
                                          "_chains_",as.character(nChains),
                                         "_thin_", as.character(thin),
                                         "_samples_", as.character(samples),
                                         ".Rdata",sep = ""))
    load(filename)
    set.seed(1)
    partition=createPartition(hM=m, nfolds=4, column="Route")
    predY = computePredictedValues(m, expected=FALSE, partition=partition,
                                   nCores = length(m$postList))
    MFCV = evaluateModelFit(hM=m, predY=predY)
    filename = file.path(MFDir, paste("model_CV_",as.character(model),
                                      c("_pa","_abundance")[modeltype],
                                      "_chains_",as.character(nChains),
                                      "_thin_", as.character(thin),
                                      "_samples_", as.character(samples),
                                      ".Rdata",sep = ""))
    save(file=filename, MFCV)
  }
}
```

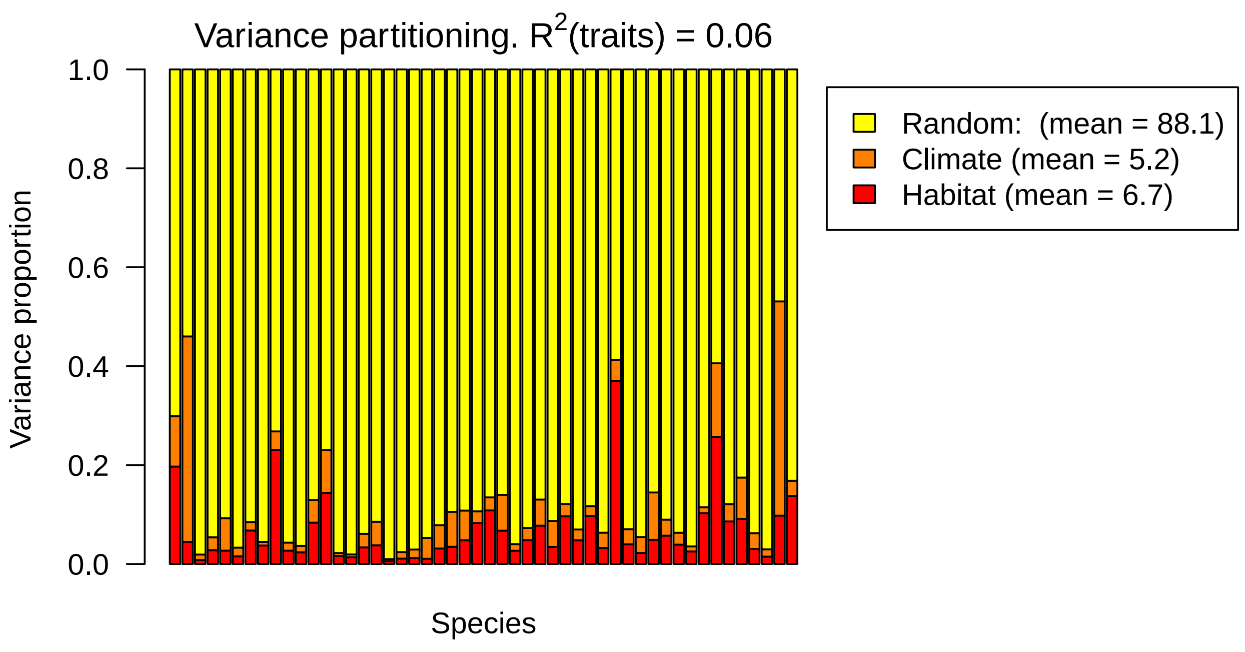

Figure 3: Variance partitioning for the spatially explicit presence/absence model with covariates

(4) Exploring parameter estimates

Compute and plot variance partitioning

The total variance explained by the model can be partitioned into the contributions of each fixed effect (or group of fixed effects) and each random effect using the `computeVariancePartitioning` function (Figure 3). This function also returns the Trait r^2, the proportion of variance explained by fixed effects explained by traits included in the model.

Visualise effects of covariates

The `plotBeta` function plots heatmaps of parameter estimates or posterior support values of species' environmental responses, i.e. how species in $\mathbf{Y}$ responds to covariates in $\mathbf{X}$ (Figure 3).

Basic parameters of `plotBeta` are

- post: Posterior summary of Beta obtained from `getPostEstimate`
- param: Controls which parameter is plotted, current options include "Beta" for parameter estimates and "Support" for posterior support.

Plots can be labeled by species and covariate names, numbers according to their order in $\mathbf{Y}$ and $\mathbf{X}$, or both.

- spNamesNumbers: Logical of length 2, where first entry controls whether species names are added to axes, and second entry controls whether species numbers are added.
- covNamesNumbers: Logical of length 2, where first entry controls whether covariate names are added to axes, and second entry controls whether covariate numbers are added.

The order in which species and covariates are plotted can also be adjusted

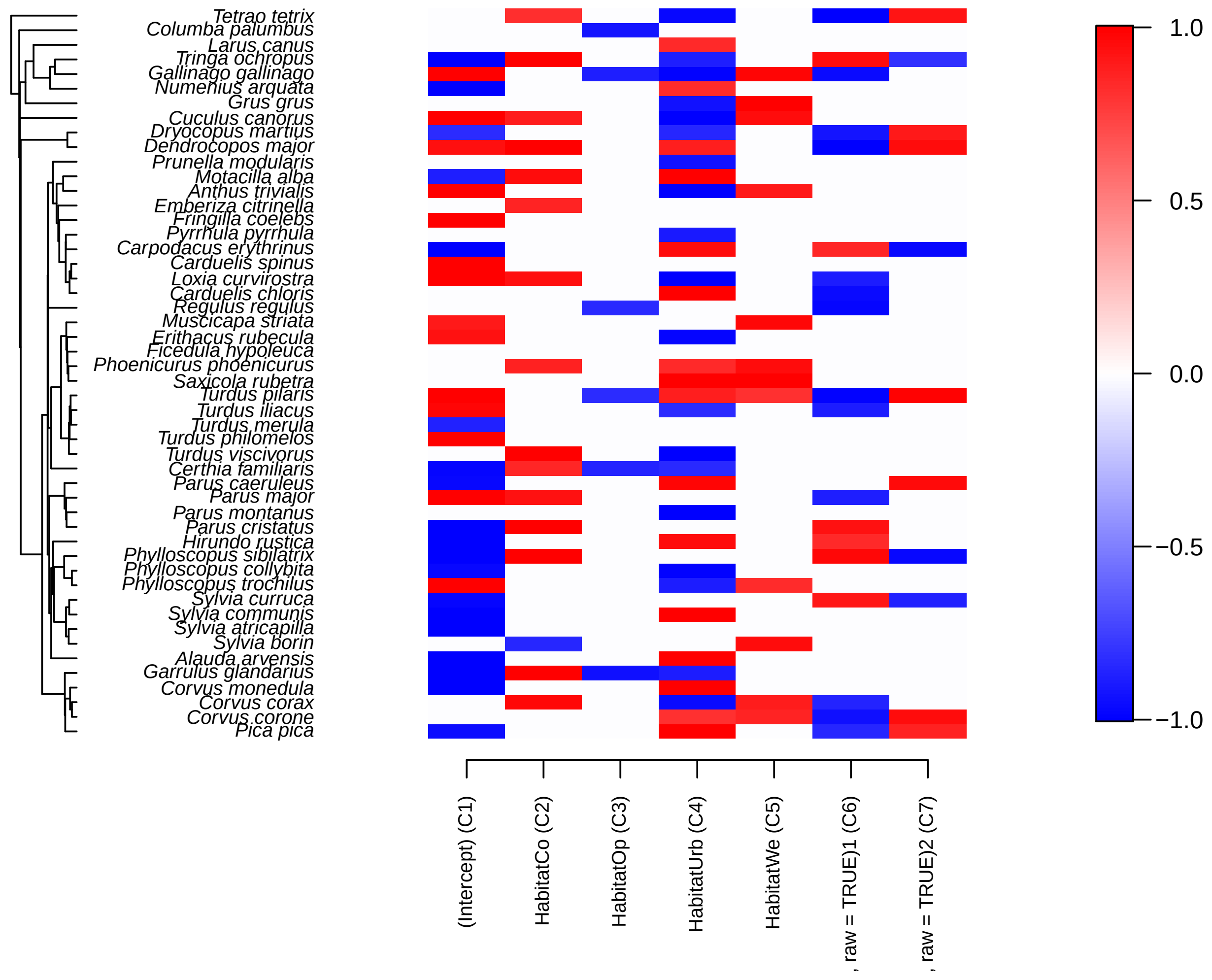

Figure 4: Posterior support values for species environmental responses

- SpeciesOrder: Controls the ordering of species, current options are “Original”, “Tree”, and “Vector”. If SpeciesOrder = “Vector”, an ordering vector must be provided (see SpVector). If plotTree = T, SpeciesOrder is ignored.
- SpVector: Controls the ordering of species if SpeciesOrder = “Vector”. If a subset of species are listed, only those will be plotted. For alphabetic ordering, try `match(1:hM$ns, as.numeric(as.factor(colnames(hM$Y))))`
- covOrder: Controls the ordering of covariates, current options are “Original” and “Vector”. If covOrder = “Vector”, an ordering vector must be provided (see covVector).
- covVector: Controls the ordering of covariates if covOrder = “Vector”. If a subset of covariates are listed, only those will be plotted.

If phylogenetic information is included in the model, we can visualize phylogenetic patterns in environmental responses by mapping them onto the phylogeny by setting `plotTree=TRUE`.

```
postBeta = getPostEstimate(m, parName="Beta")
plotBeta(m, post=postBeta, param="Support", plotTree=TRUE, spNamesNumbers=c(TRUE,FALSE))
```

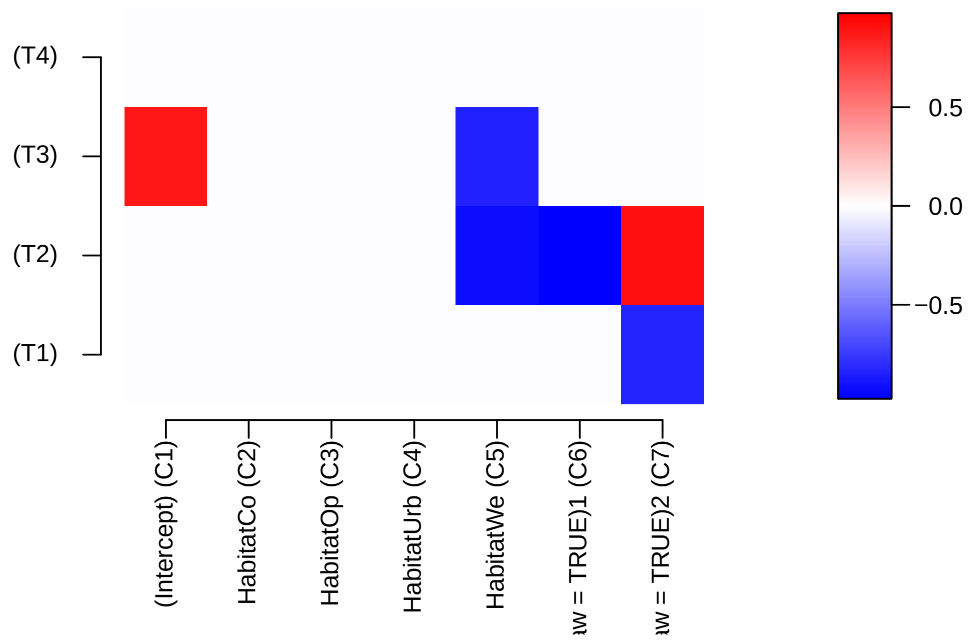

Figure 5: Posterior support values for effects of traits on species' environmental responses

Visualise effects of traits

Similarly, the `plotGamma` function plots heatmaps of parameter estimates or posterior support values of the effect of traits on species' environmental responses, i.e. how species' environmental responses (β) responds to traits in **T** (Figure 4).

```
postGamma = getPostEstimate(m, parName="Gamma")
plotGamma(m, post=postGamma, param="Support", trNamesNumbers=c(TRUE,TRUE))
```

Plot species associations

As advantage of HMSC and other joint species distribution models is that they allow estimating residual covariances among species, after controlling for shared responses to the covariates. In `HMSC` these associations are given by the Ω matrix, and can be extracted using the `computeAssociations` function. For visual clarity, we here plot only those associations with at least 95% posterior support, and order species into groups associated positively and negatively (Figure 6).

```
OmegaCor = computeAssociations(m)
```

```
supportLevel = 0.95
for (r in 1:m$nr){
  plotOrder = corrMatOrder(OmegaCor[[r]]$mean,order="AOE")
  toPlot = ((OmegaCor[[r]]$support>supportLevel) +
           (OmegaCor[[r]]$support<(1-supportLevel))>0)*OmegaCor[[r]]$mean
  par(xpd=T)
  colnames(toPlot)=rownames(toPlot)=gsub("_"," ",x=colnames(toPlot))
  corrplot(toPlot[plotOrder,plotOrder], method = "color",
           col=colorRampPalette(c("blue","white","red"))(200),
            title="",type="lower",tl.col="black",tl.cex=.7, mar=c(0,0,6,0))
}
```

Assessing parameter estimates numerically

For the univariate paramters α and ρ, we can assess parameter estimates and their associated uncertainty by looking at the mean, standard deviations and quantiles of their posterior distributions (Table), returned by `summary()`.

```
mpost = convertToCodaObject(m)
print(summary(mpost$Rho))
summary(mpost$Alpha[[1]])
```

Table 1: Mean, SD, SE, and quantiles for the phylogenetic signal parameter Rho and the spatial scale parameter Alpha for the first five spatial latent factors

	Rho	Alpha1[factor1]	Alpha1[factor2]	Alpha1[factor3]	Alpha1[factor4]	Alpha1[factor5]
Mean	0.0909	341.75	32.46	0.76	227.69	99.54
SD	0.1477	102.77	11.58	3.57	63.85	88.76
Naive SE	0.0023	1.62	0.18	0.06	1.01	1.40
Time-series SE	0.0023	4.85	0.36	0.07	1.13	1.61
2.5%	0.0000	203.42	12.71	0.00	127.14	38.14
25%	0.0000	266.99	25.43	0.00	177.99	50.86
50%	0.0000	317.85	25.43	0.00	216.14	76.28
75%	0.1600	394.13	38.14	0.00	266.99	101.71
97.5%	0.4800	597.55	63.57	12.71	381.42	343.59

(5) Making predictions

Visualize predicted variation over environmental gradients

An important feature of the HMSC model is the ability to predict community composition along environmental gradients described by the covariates included in the model (**X** matrix). The `constructGradient` function is built for this purpose. `constructGradient` takes as arguments a named focal variable, and a list of non-focal variables. For each non-focal variable, details must be given about how the variable is to be treated, as list(type,value), where value is needed only if type = 3.

- type = 1 fixes to the most likely value (defined as expected value for covariates, mode for factors)
- type = 2 fixes to most likely value, given the value of focal variable, based on a linear relationship
- type = 3 fixes to the value given

If a non-focal variable is not listed, type = 2 is used as default.

Fixing non-focal variables to the most likely value (type = 1) provides predictions for the marginal effect of the focal variable, that is the effect of the focal variable independently of all non-focal variables. In contrast, letting non-focal variables covary with the focal variable (type = 2) provides predictions for the net effect of the focal variable, that is the total effect of the focal variable and any non-focal variables covarying with the focal variable. Note that if the focal variable is continuous, selecting type 2 for a non-focal categorical variable can cause abrupt changes in the predicted response. Fixing non-focal variables (type = 3) can be useful e.g. for generating predictions for concrete ecological scenarios.

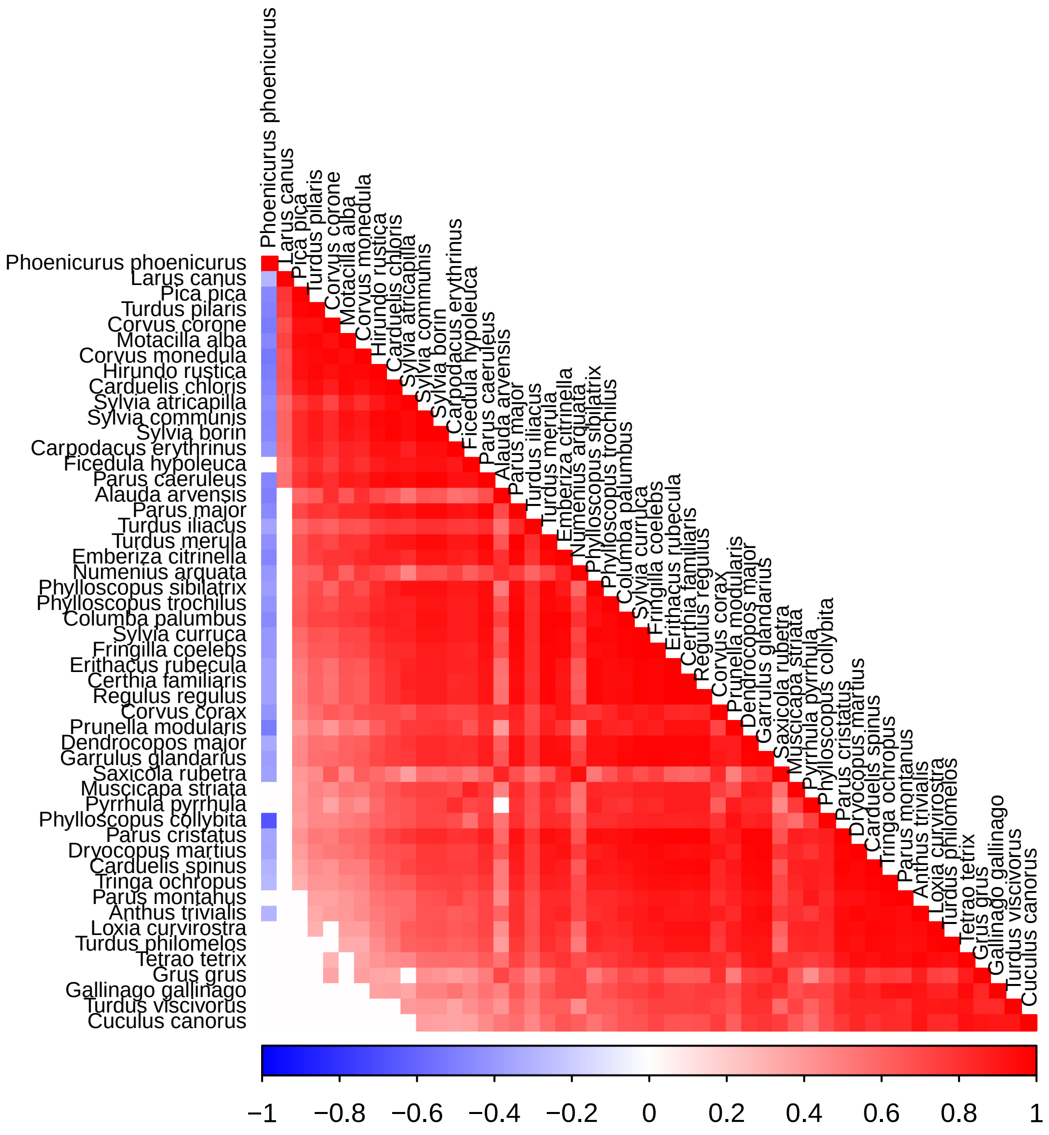

Figure 6: Species associations

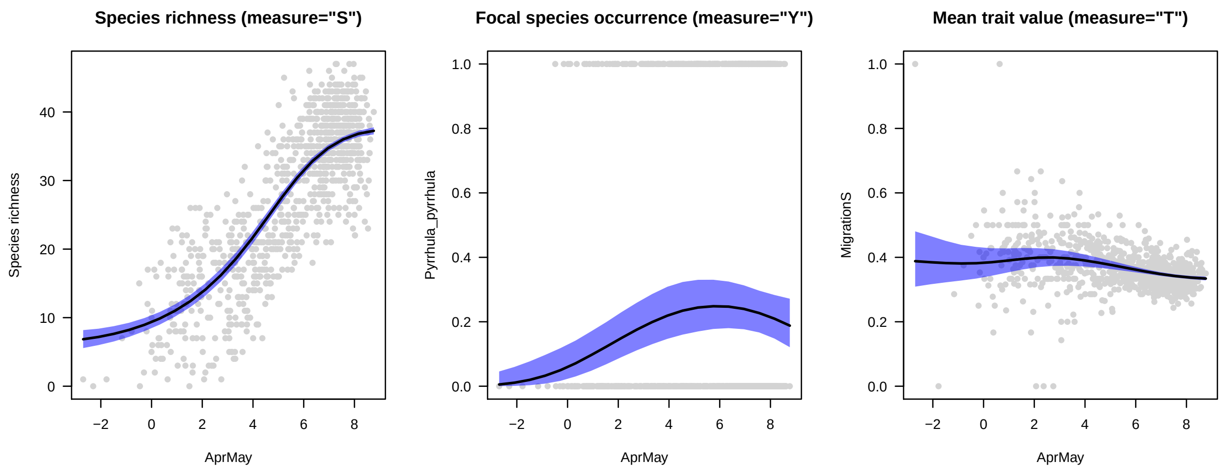

Figure 7: Predicted values along a continuous gradient

As an example, we construct a gradient over variation in AprMay, with Habitat fixed to urban (Figure 7), and use the `predict`function to obtain predictions along the gradient.

```
thin = 100
samples = 1000
nChains = 4
model = 2
modeltype = 1
filename = file.path(ModelDir, paste("model_",as.character(model),
                                    c("_pa","_abundance")[modeltype],
                                    "_chains_",as.character(nChains),
                                    "_thin_", as.character(thin),
                                    "_samples_", as.character(samples),
                                    ".Rdata",sep = ""))
load(filename)

Gradient = constructGradient(m, focalVariable="AprMay",
                             non.focalVariables=list(Habitat=list(3,"Urb")))

predY = predict(m, XData=Gradient$XDataNew, studyDesign=Gradient$studyDesignNew,
                ranLevels=Gradient$rLNew, expected=TRUE)
```

The `plotGradient` function produces plots of predicted values along the gradient.

- measure = “S” plots species richness
- measure = “Y” plots the response of a species given by index
- measure = “T” plots community-weighed mean values of traits given by index

```
par(mfrow=c(1,3))
plotGradient(m, Gradient, pred=predY, measure="S", las=1,
             showData = TRUE, main='Species richness (measure="S")')
plotGradient(m, Gradient, pred=predY, measure="Y", index=40, las=1,
              showData = TRUE, main='Focal species occurrence (measure="Y")')
plotGradient(m, Gradient, pred=predY, measure="T", index=3, las=1,
              showData = TRUE, main='Mean trait value (measure="T")')
```

Visualize spatial variation

For spatially explicit analyses, it can be interesting to visualize spatial patterns in the model estimates as maps showing e.g. predicted species occurrence (Figure 8), or predicted species richness (Figure 9).

The following part of the code predicts the species occurrence matrix (**predYR**) and extracts xy-coordinates (xy).

```r
# READING THE GRID DATA
grid = read.csv(file.path(dataDir, "grid_10000.csv"))
grid = grid[!(grid$Habitat=="Ma"),]

# DEFINING THE NEW STUDY DESIGN
nyNew = nrow(grid)
StudyDesignNew = matrix(NA,nyNew,2, dimnames=list(NULL,names(m$studyDesign)))
StudyDesignNew[,1] = sprintf('new_Route_%.3d',1:nyNew)
StudyDesignNew[,2] = sprintf('new_Year_%.3d',1:nyNew)
StudyDesignNew = as.data.frame(StudyDesignNew)
StudyDesignAll=rbind(m$studyDesign,StudyDesignNew)

# DEFINING RANDOM EFFECTS THAT INCLUDE BOTH OLD (USED FOR MODEL FITTING)
# AND NEW (THE GRID DATA) UNITS
rL1 = m$rL[[1]]
xyold = rL1$s
xy = grid[,1:2]
rownames(xy) = StudyDesignNew[,1]
colnames(xy) = colnames(xyold)
xyall = rbind(xyold,xy)
rL1$pi = StudyDesignAll[,1]
rL1$s = xyall
if(FALSE){
  predYR1 = predict(m, studyDesign=StudyDesignNew, XData=grid, ranLevels=list(Route=rL1),
                    expected=TRUE, predictEtaMean=TRUE)
  predYR = apply(abind(predYR1,along=3),c(1,2),mean)
  save(predYR,file="panels/predictions/predYR_thin_100_samples_1000_grid_10000.Rdata")
} else {
  load("panels/predictions/predYR_thin_100_samples_1000_grid_10000.Rdata")
  ta = 1:dim(predYR)[1]
  predYR = predYR[ta,]
  xy = grid[,1:2]
  xy = xy[ta,]
}

# COMPUTE SPECIES RICHNESS (S), COMMUNITY WEIGHTED MEANS (predT),
# REGIONS OF COMMON PROFILE (RCP)
S=rowSums(predYR)
predT = (predYR%*%m$Tr)/matrix(rep(S,m$nt),ncol=m$nt)
RCP = kmeans(predYR, 7)
RCP$cluster = as.factor(RCP$cluster)

# EXTRACT THE OCCURRENCE PROBABILITIES OF ONE EXAMPLE SPECIES
pred_Cm = predYR[,50]

# MAKE A DATAFRAME OF THE DATA TO BE PLOTTED
mapData=data.frame(xy,S,predT,pred_Cm,RCP$cluster)
```

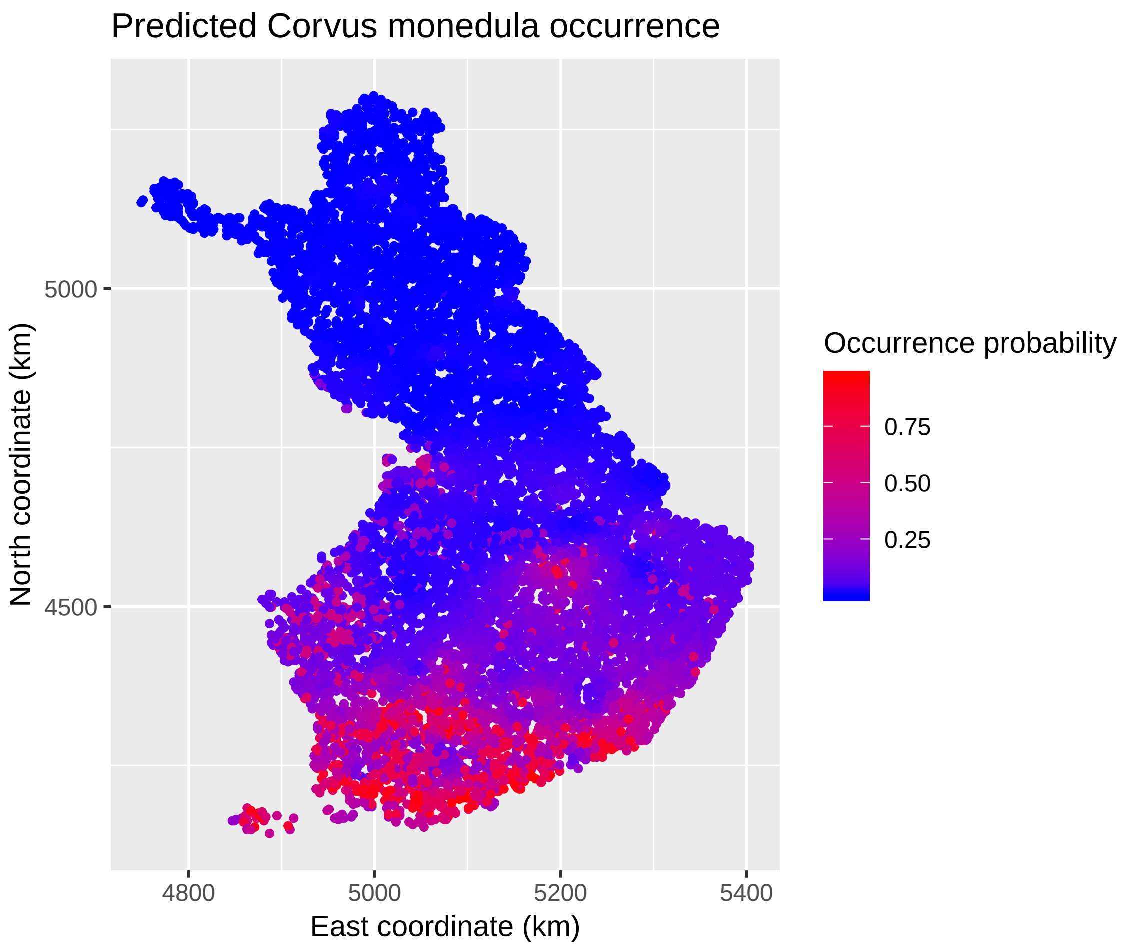

Figure 8: Predicted occurrence probability of *Corvus monedula*

We are now ready to map the predicted values onto the map of Finland:

```r
sp <- ggplot(data = mapData, aes(x=x, y=y, color=pred_Cm))+geom_point(size=1)
sp + ggtitle("Predicted Corvus monedula occurrence") +
  xlab("East coordinate (km)") + ylab("North coordinate (km)") +
  scale_color_gradient(low="blue", high="red", name ="Occurrence probability")
```

```r
sp <- ggplot(data = mapData, aes(x=x, y=y, color=S))+geom_point(size=1)
sp + ggtitle("Predicted species richness") +
  xlab("East coordinate (km)") + ylab("North coordinate (km)") +
  scale_color_gradient(low="blue", high="red", name ="Species richness")
```

References

Ovaskainen, O., D. B. Roy, R. Fox, and B. J. Anderson. 2016. "Uncovering Hidden Spatial Structure in Species Communities with Spatially Explicit Joint Species Distribution Models." *Methods in Ecology and Evolution* 7 (4): 428–36. doi:10.1111/2041-210x.12502.

Ovaskainen, O., G. Tikhonov, A. Norberg, F. Guillaume Blanchet, L. Duan, D. Dunson, T. Roslin, and N. Abrego. 2017. "How to Make More Out of Community Data? A Conceptual Framework and Its

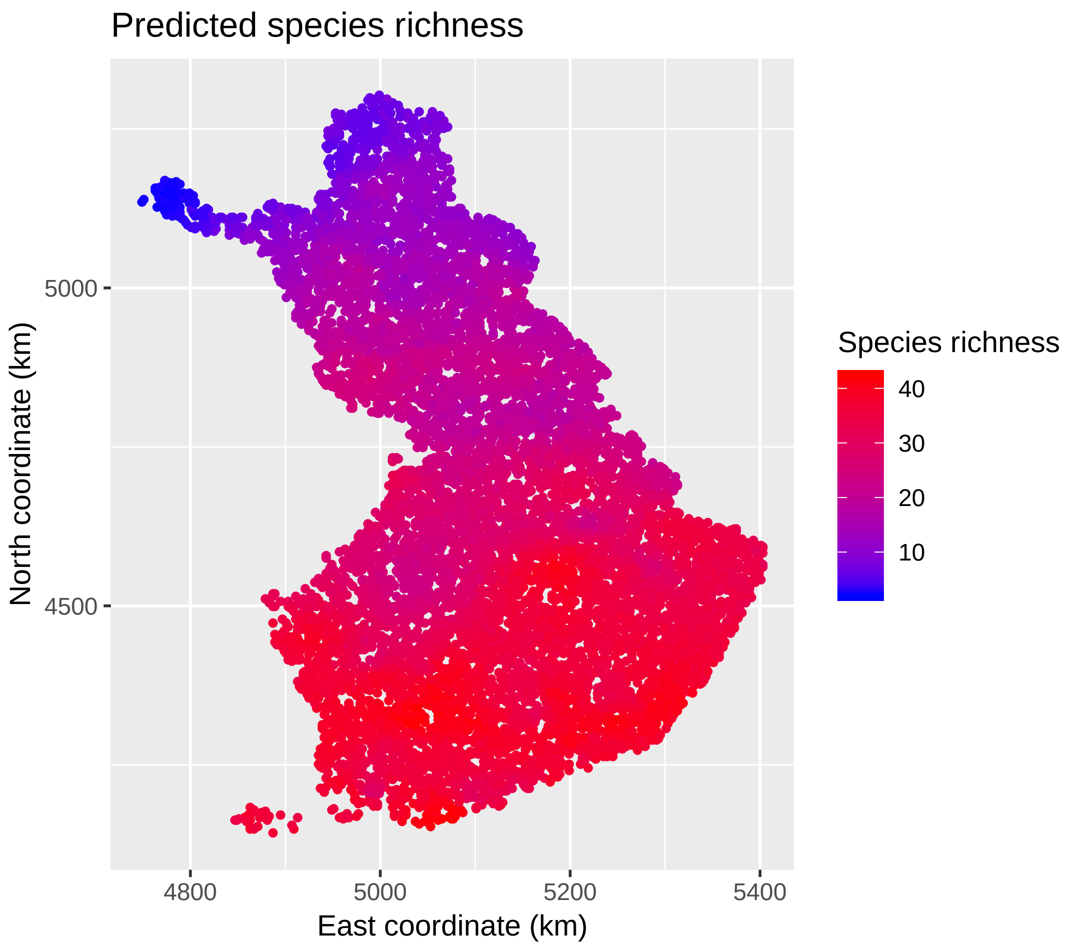

Figure 9: Predicted species richness

Implementation as Models and Software." *Ecol Lett* 20: 561–76. doi:10.1111/ele.12757.