Processing and Analysis of Biological Data

Joint Species Distribution Modelling

Øystein H. Opedal

14 Dec 2022

Introduction

In community ecology, chemical ecology and many other sub-fields, data are often multivariate in comprising e.g. many species or many chemical compounds. Traditionally, these data have normally been analysed by ordination methods such as principal component analysis, (detrended) correspondence analysis, redundancy analysis or non-metric multidimensional scaling (NMDS). Recently however, there has been an explosive development of model-based approaches falling into the category of "joint models". In this chapter we will consider such joint models applied to multivariate data on species communities, thus fitting *joint species distribution models*.

The HMSC framework

Hierarchical Modelling of Species Communities (Hmsc) is a conceptual framework for analysis of community-ecological data, currently implemented in the Bayesian framework in the `R` package `Hmsc`. For a complete introduction to the framework, have a look at the paper

Ovaskainen, O., G. Tikhonov, A. Norberg, F. G. Blanchet, L. Duan, D. Dunson, T. Roslin, and N. Abrego. 2017. How to make more out of community data? A conceptual framework and its implementation as models and software. Ecol Lett 20:561-576.

In the following sections, we will go through the basics of model fitting and interpretation of an `Hmsc` model. As an example, we will analyse a dataset on Euglossine bee communities from the Brazilian atlantic forest.

Model fitting

When fitting models with Bayesian inference, the model fitting can take a long time and is always performed separately from subsequent postprocessing. First, we read in and format the data files.

```r
Y = read.csv(file="datasets/euglossini/Y.csv")
XData = read.csv(file="datasets/euglossini/XData.csv")
XData$method = as.factor(XData$method)

TrData = read.csv(file="datasets/euglossini/TrData.csv")
TrData = as.data.frame(TrData)
rownames(TrData) = colnames(Y)
names(TrData) = "genus"

dfPi = read.csv(file="datasets/euglossini/dfPi.csv")
dfPi$SA = as.factor(dfPi$SA)
dfPi$SU = as.factor(dfPi$SU)
```

After reading the data, it is good to check that all the data comonents have the same number of samples and species. For example, the predictor variables (`XData`) should have as many rows as the response variables (`Y`).

```
dim(Y)
```

```
## [1] 178  61
```

```
dim(TrData)
```

```
## [1] 61  1
```

```
dim(XData)
```

```
## [1] 178   9
```

```
dim(dfPi)
```

```
## [1] 178   2
```

Next, we proceed to set up our model. In the current model we will fit two hierarchical random factors, study area (SA) and sampling unit (SU).

```
library(Hmsc)
```

```
## Loading required package: coda
```

```
rL1 = HmscRandomLevel(units = unique(dfPi$SA))
rL2 = HmscRandomLevel(units = unique(dfPi$SU))
rL1
```

```
## Hmsc random level object with 72 units. Spatial dimensionality is 0 and number of covariates is 0.
```

```
rL2
```

```
## Hmsc random level object with 178 units. Spatial dimensionality is 0 and number of covariates is 0.
```

Next, we set the formulae for the predictors. We also include a "trait" genus, which tells us which genus the species belong to. This will act as a very simple phylogenetic structure in the model. (`Hmsc` also allows using an actual phylogeny, but we skip that here for simplicity).

```
XFormula = ~ method + effort + altitude + MAT + MAP + Tseason + Pseason +
              forest. + lu_het

TrFormula= ~genus
```

Now, we are ready to set up the model object. We will fit a presence-absence model, which means that we only ask whether a species is present (1) or absent (0) in each sample. We therefore choose a binomial model with a *probit* link function (recall that the probit link is very similar to the logit link, but works better in the context of Bayesian model fitting).

```
m1 = Hmsc(Y = as.matrix((Y>0)*1),
          XData = XData,  XFormula = XFormula,
          TrData = TrData, TrFormula = TrFormula,
          distr = "probit",
          studyDesign = dfPi,
          ranLevels = list(SA=rL1, SU=rL2))
```

The next step is to set the sampling parameters and run the MCMC sampling.

```
thin = 10
samples = 1000
adaptNf = ceiling(0.4*samples*thin)
transient = ceiling(0.5*samples*thin)
nChains = 2

a=Sys.time()
m1 = sampleMcmc(m1, samples = samples, thin = thin,
                adaptNf = rep(adaptNf, 2),
                transient = transient,
                nChains = nChains, nParallel = 2, updater=list(GammaEta=FALSE))
Sys.time()-a

filename = paste0("model_thin_", thin, ".RData")
save(m1, file=filename)
```

Postprocessing

We are now ready to start exploring the fitted model. First, we read the model object and explore convergence by computing effective sample sizes and potential scale reduction factors.

```
load("model_thin_10.RData")

mpost = convertToCodaObject(m1)

esBeta = effectiveSize(mpost$Beta)
summary(esBeta)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   103.8   395.8   578.7   614.5   763.7  1699.5
```

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

```
##    Point est.       Upper C.I.
##  Min.   :0.9994   Min.   :0.9995
##  1st Qu.:1.0052   1st Qu.:1.0177
##  Median :1.0163   Median :1.0685
##  Mean   :1.0695   Mean   :1.2424
##  3rd Qu.:1.0570   3rd Qu.:1.2245
##  Max.   :2.4207   Max.   :5.5202
```

Next, we assess the explanatory power of the model.

```
predY = computePredictedValues(m1, expected=T)
MF = evaluateModelFit(hM=m1, predY = predY)

mean(MF$TjurR2, na.rm=T)
```

```
## [1] 0.4741199
```

```
mean(MF$AUC, na.rm=T)
```

```
## [1] 0.9674697
```

```
range(MF$TjurR2)
```

```
## [1] 0.09148615 0.86626990
```

```
range(MF$AUC)
```

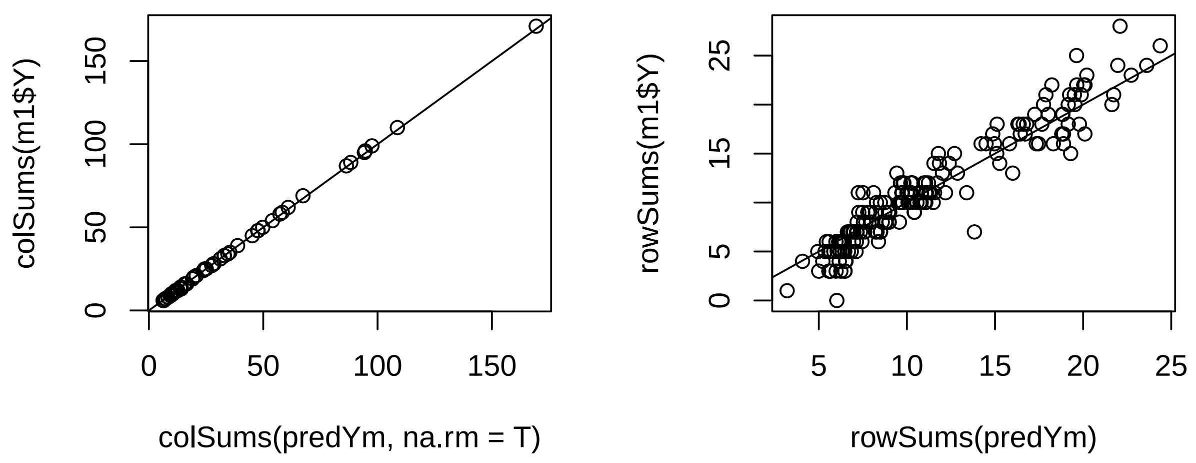
```
## [1] 0.8608059 1.0000000
```

We can also check if the model predicts well the number of species per site (species richness), and the number of occurrences per species. These are given by the row and column sums of the matrix `Y`.

```
predYm = apply(predY, 1:2, mean)

par(mfrow=c(1,2))
plot(colSums(predYm,na.rm=T), colSums(m1$Y))
lines(-1000:1000, -1000:1000)
#cor(colSums(predYm), colSums(m1$Y))^2

plot(rowSums(predYm), rowSums(m1$Y))
lines(-1000:1000, -1000:1000)
```



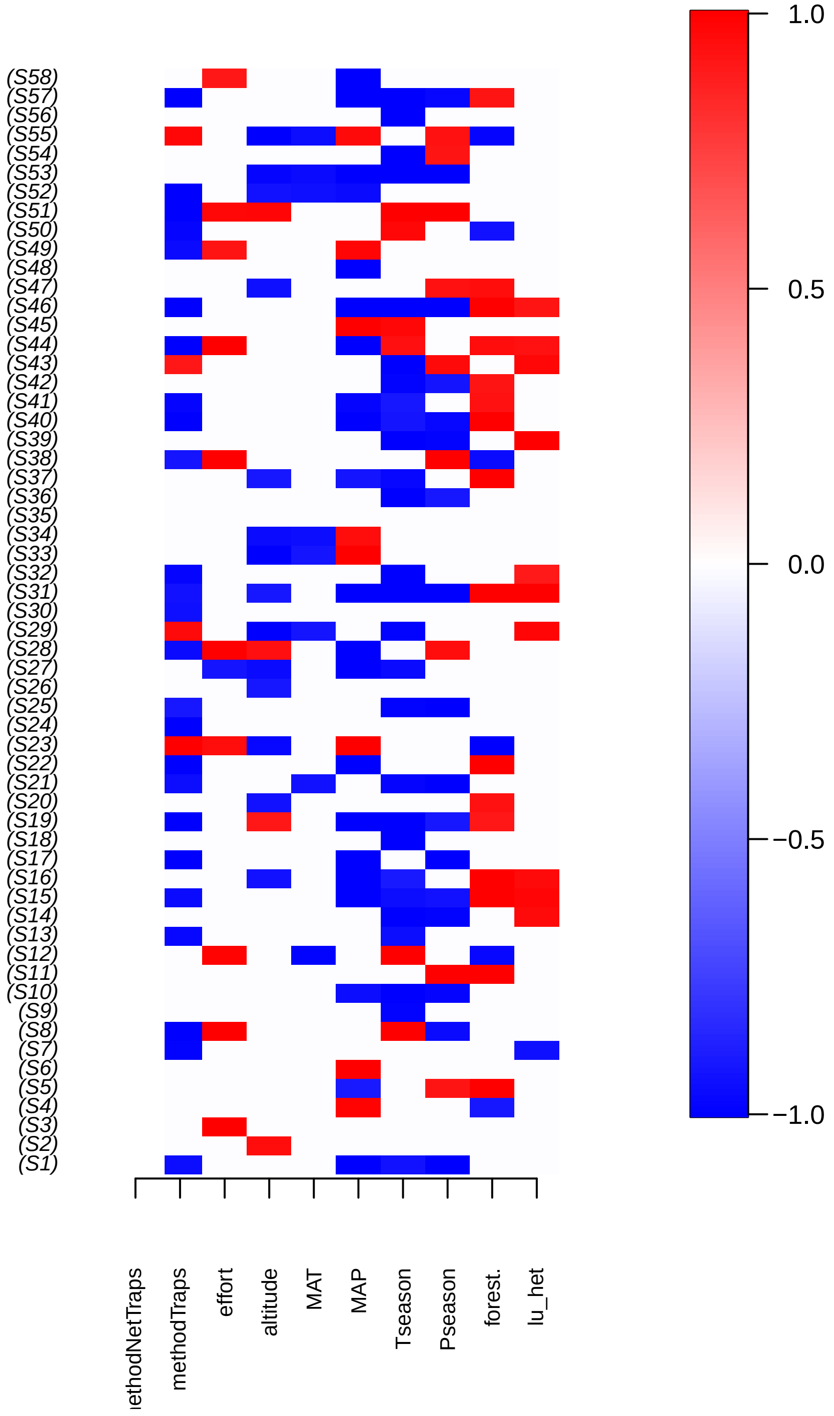
```
#cor(rowSums(predYm), rowSums(m1$Y))^2
```

Extracting parameter estimates

The HMSC model includes a lot of parameters, and it rarely makes sense to report all of them in a table (at least not in the main text). One option is to plot the matrix of parameter estimates (or their support, as in the example below).

```
pbeta = getPostEstimate(m1, "Beta")

plotBeta(m1, post=pbeta, param = "Support", covOrder="Vector", covVector=c(2:11), supportLevel = .95,
         SpeciesOrder = "Vector", SpVector = 1:58, spNamesNumbers=c(F,T), covNamesNumbers = c(T,F))
```

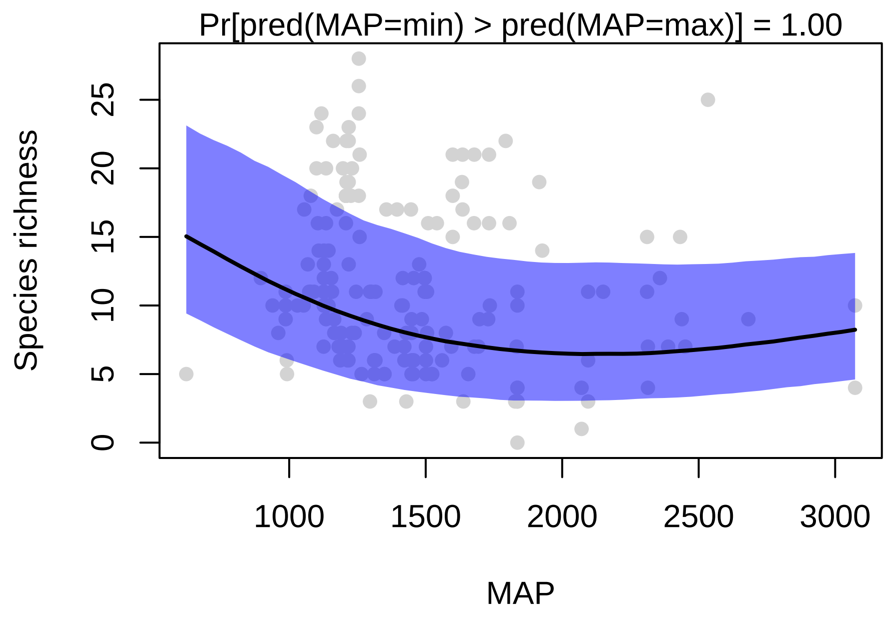


Because there are so many parameters, it can also be useful to summarize the results across species by plotting e.g. a covariate against the predicted species richness.

```
x=2
Gradient=constructGradient(m1, focalVariable = "MAP", ngrid=50,
                                   non.focalVariables = list(
                                     method = list(1),
                                     effort = list(1),
                                     altitude = list(x),
                                     MAT = list(x),
                                     Tseason = list(x),
                                     Pseason = list(x),
                                     forest = list(x),
                                     lu_het = list(x)))

predY = predict(m1, Gradient=Gradient, expected=TRUE, predictEtaMean=FALSE)

plotGradient(m1, Gradient, predY, measure="S", showData = T)
```



```
## [1] 0.003
```