HMSC-R 3.0: Getting started with HMSC-R: spatial models

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

26 August 2019

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-R 3.0: Getting started with HMSC-R: univariate models" and "HMSC-R 3.0: Getting started with HMSC-R: low-dimensional multivariate models". In the first vignette we shortly discussed how to fit spatially explicit models to the univariate case. Here, we continue to demonstrate how to use HMSC to make spatially explicit models for the multivariate case and for large datasets.

To get HMSC-R in use, you need to first install it [https://github.com/hmsc-r/HMSC] and then load it.

```
library(Hmsc)
library(corrplot)
library(MASS)
set.seed(6)
```

We also set the random number seed to make the results presented here reproducible.

Generating simulated data

To illustrate how to use spatial models in Hmsc, we generate data for 5 species (`ns`) on 100 sampling units (`n`). We include only one environmental predictor (`x1`) and give the true intercept (`alpha`) and slope (`beta1`) parameters to construct the matrix of the linear predictors. So far, this is similar to what we did in the low-dimensional multivariate case, but note that we now call this linear predictor `Lf` instead of `L`. This indicates that this is the fixed effects part of the linear predictor, i.e. the effect that can be explained by the environmental covariates. In addition to the effect from the environmental covariates, we now add spatially structured residuals, indicated by `Lr`. To generate these spatial residuals, we first simulate some random x and y coordinates for all sampling units (`xycoords`). We then generate a spatial predictor (`eta1`) using an exponentially decreasing spatial covariance function where we have set the spatial scale parameter (`alpha`) to 0.35. Next, we set the true slope parameters (`lambda1`) for the species responses to the spatial predictor and compute the spatial residuals for each species in `Lr`.

```
n = 100
ns = 5
beta1 = c(-2,-1,0,1,2)
alpha = rep(0,ns)
beta = cbind(alpha,beta1)
sigma = rep(1,ns)
x1 = rnorm(n)
x = cbind(rep(1,n),x1)
Lf = x%*%t(beta)
```

```
xycoords = matrix(runif(2*n),ncol=2)
colnames(xycoords) = c("x-coordinate","y-coordinate")
rownames(xycoords) = 1:n

sigma.spatial = c(2)
alpha.spatial = c(0.35)
Sigma = sigma.spatial^2*exp(-as.matrix(dist(xycoords))/alpha.spatial)
eta1 = mvrnorm(mu=rep(0,n), Sigma=Sigma)
lambda1 = c(1,2,-2,-1,0)
Lr = eta1%*%t(lambda1)
L = Lf + Lr
y = as.matrix(L + matrix(rnorm(n*ns),ncol=ns))
yprob = 1*((L +matrix(rnorm(n*ns),ncol=ns))>0)
XData = data.frame(x1=x[,2])
```

We can now visualize the species response matrix y as function of the x and y coordinates (Fig. 1). This shows that indeed nearby sampling units have similar responses for species that have a non-zero loading (lambda) to the spatially structured latent variable.

```
rbPal = colorRampPalette(c('cyan','red'))
par(mfrow=c(2,3))
Col = rbPal(10)[as.numeric(cut(x[,2],breaks = 10))]
plot(xycoords[,2],xycoords[,1],pch = 20,col = Col,main=paste('x'))
for(s in 1:ns){
  Col = rbPal(10)[as.numeric(cut(y[,s],breaks = 10))]
  plot(xycoords[,2],xycoords[,1],pch = 20,col = Col,main=paste('Species',s))
}
```

A spatially explicit model in Hmsc

To fit a spatially explicit model with Hmsc, we construct the random effect using the `sData` input argument where we give the coordinates of the sampling units.

```
studyDesign = data.frame(sample = as.factor(1:n))
rL.spatial = HmscRandomLevel(sData = xycoords)
rL.spatial = setPriors(rL.spatial,nfMin=1,nfMax=1) #We limit the model to two latent variables for visu
m.spatial = Hmsc(Y=yprob, XData=XData, XFormula=~x1,
studyDesign=studyDesign, ranLevels=list("sample"=rL.spatial),distr="probit")
```

Model fitting and evaluation of explanatory and predictive power can be done as before. We first set the MCMC sampling parameters.

```
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
```

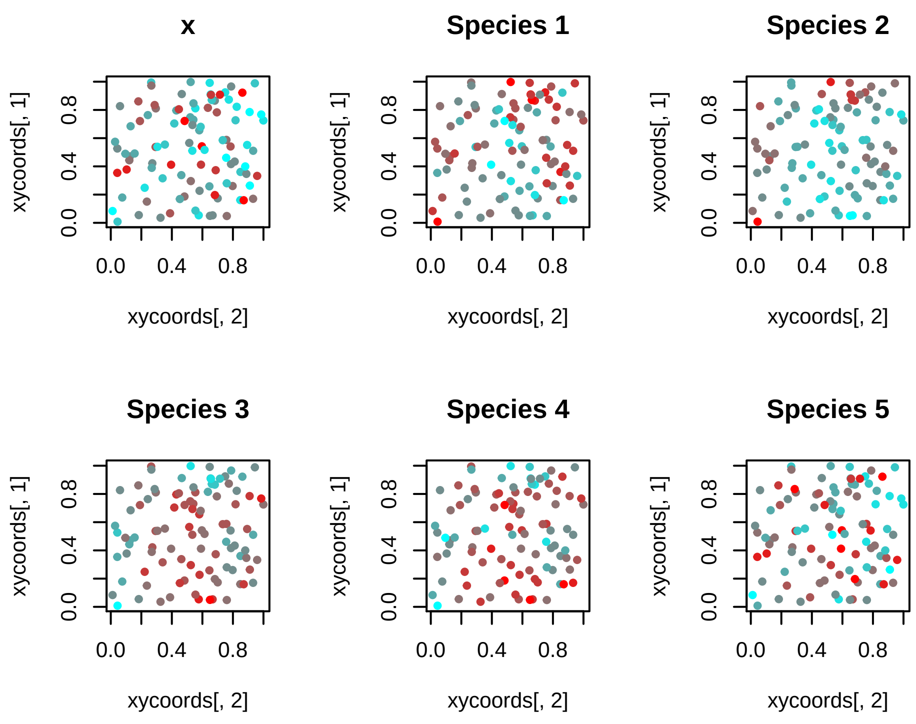

Figure 1: Plots of simulated spatially structured data.

```
} else {
   # with this option, the vignette evaluates slow but it reproduces the results of
   # the .pdf version
   thin = 10
   samples = 1000
   transient = 1000
   verbose = 0
}

m.spatial = sampleMcmc(m.spatial, thin = thin, samples = samples, transient = transient,
               nChains = nChains, verbose = verbose,updater=list(GammaEta=FALSE))
```

```
## [1] "Computing chain 1"
## [1] "Computing chain 2"
```

The explanatory and predictive power of the model can now be calculated in the same way as before.

```
#Explanatory power
preds.spatial = computePredictedValues(m.spatial)
MF.spatial = evaluateModelFit(hM=m.spatial, predY=preds.spatial)
MF.spatial
```

```
## $RMSE
## [1] 0.2919539 0.2012685 0.2626055 0.2650880 0.3012978
##
## $AUC
## [1] 0.9543468 0.9888393 0.9581857 0.9637500 0.9504831
```

```
##
## $TjurR2
## [1] 0.5057258 0.5774853 0.4108343 0.4656761 0.6187248
```

```
#Predictive power
partition = createPartition(m.spatial, nfolds = 2, column = "sample")
cvpreds.spatial = computePredictedValues(m.spatial, partition=partition,updater=list(GammaEta=FALSE))
```

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

```
cvMF.spatial = evaluateModelFit(hM=m.spatial, predY=cvpreds.spatial)
cvMF.spatial
```

```
## $RMSE
## [1] 0.3795827 0.3242206 0.3583916 0.3851559 0.3060093
##
## $AUC
## [1] 0.8547839 0.8325893 0.7037562 0.7300000 0.9488728
##
## $TjurR2
## [1] 0.2972472 0.1849972 0.1130621 0.1051705 0.5902591
```

As the model includes a spatially structured random effect, its predictive power is based on both the fixed and the random effects. Concerning the latter, the model can utilize observed data from nearby sampling units included in model fitting when predicting the response for a focal sampling unit that is not included in model fitting.

The estimated spatial scale of the random effect is given by the parameter `Alpha[[1]]`. Let's have a look at the MCMC trace plot for this parameter (Fig. 3).

```
mpost.spatial = convertToCodaObject(m.spatial)
plot(mpost.spatial$Alpha[[1]])
```

```
summary(mpost.spatial$Alpha[[1]])
```

```
##
## Iterations = 1010:11000
## Thinning interval = 10
## Number of chains = 2
## Sample size per chain = 1000
##
## 1. Empirical mean and standard deviation for each variable,
##    plus standard error of the mean:
##
##          Mean             SD       Naive SE Time-series SE
##      0.336384       0.143647       0.003212       0.007340
```

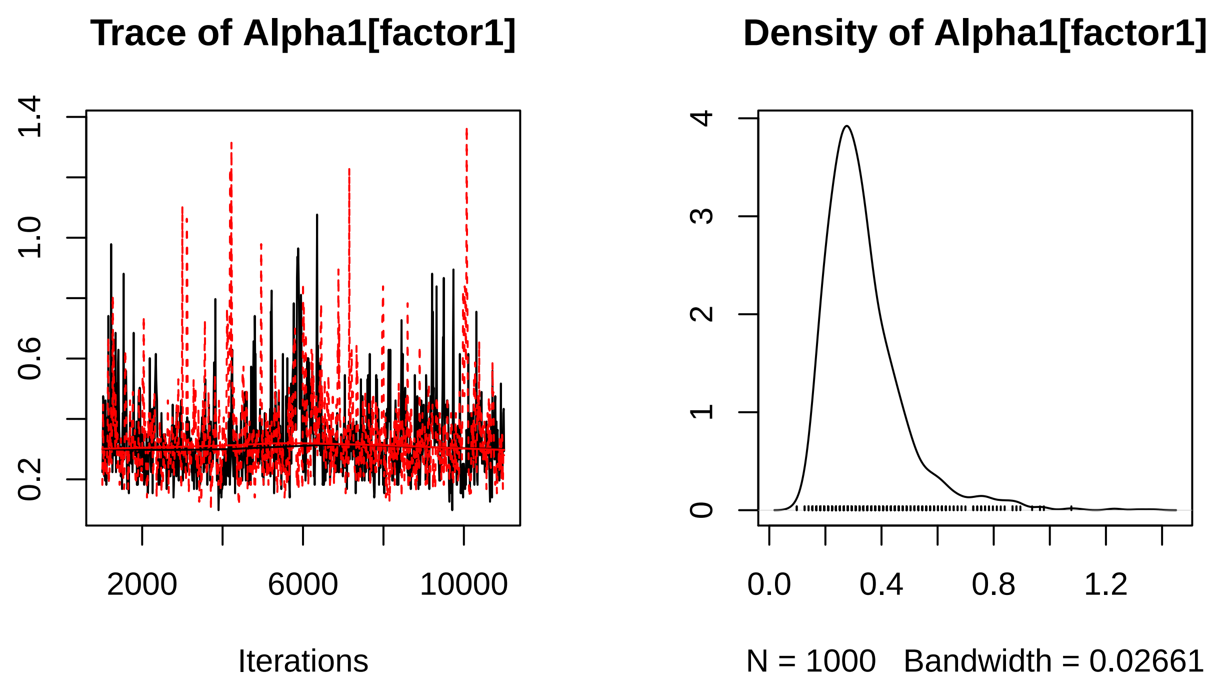

Figure 2: Posterior trace plot of the spatial scale parameter of the spatial model.

```
##
## 2. Quantiles for each variable:
##
##   2.5%    25%    50%    75%  97.5%
## 0.1674 0.2377 0.3076 0.3915 0.7411
```

For comparison, let us fit a non-spatial model to the same data.

```
m = Hmsc(Y=yprob, XData=XData, XFormula=~x1, studyDesign = studyDesign, distr="probit")
m = sampleMcmc(m, thin = thin, samples = samples, transient = transient,
               nChains = nChains, verbose = verbose)
```

```
## [1] "Computing chain 1"
## [1] "Computing chain 2"
```

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

```
## $RMSE
## [1] 0.3827911 0.3595311 0.3739090 0.3915828 0.3018607
##
## $AUC
## [1] 0.8334143 0.6495536 0.5726435 0.6312500 0.9504831
##
## $TjurR2
## [1] 0.285287389 0.040537987 0.008576347 0.038951856 0.613456722
```

```r
partition = createPartition(m, nfolds = 2, column = "sample")
preds = computePredictedValues(m, partition=partition)
```

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

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

```
## $RMSE
## [1] 0.3979680 0.4042271 0.3866059 0.4083913 0.3038340
##
## $AUC
## [1] 0.8042739 0.5833333 0.5457123 0.5268750 0.9496779
##
## $TjurR2
## [1]  0.25805609 -0.04247210 -0.01109638  0.01487474  0.58201342
```

We observe that both the explanatory and the predictive power is lower than for the model with a spatial random effect for those species that had spatially structured respones.

Spatial models for big datasets

For large datasets, i.e. those with $>$ 1000 sampling units, the standard spatial models as described above may become computationally infeasible. For such datasets, we implemented two alternative approaches to account for the spatial structure in the data: the 'Nearest Neighbour Gaussian Process (NNGP)' and the 'Gaussian Predictive Process (GPP)'. The detailes of this method are described in Tikhonov et al. (n.d.).

NNGP models

If we want to fit a NNGP model we have to set `sMethod` to 'NNGP' when constructing the random level. Additionally, we can specify how many neighbours we want to use by setting the `nNeighbours` argument. When we do not explicitely set `nNeighbours` when constructing the random level, this parameters is set to 10 as a standard.

```r
rL.nngp = HmscRandomLevel(sData = xycoords, sMethod = 'NNGP', nNeighbours = 10)
rL.nngp = setPriors(rL.nngp,nfMin=1,nfMax=1)
```

Running the model and checking the fit is done in the same way as earlier.

```r
m.nngp = Hmsc(Y=yprob, XData=XData, XFormula=~x1,
         studyDesign=studyDesign, ranLevels=list("sample"=rL.nngp),distr="probit")
m.nngp = sampleMcmc(m.nngp, thin = thin, samples = samples, transient = transient,
                nChains = nChains, verbose = verbose, updater=list(GammaEta=FALSE))
```

```
## [1] "Computing chain 1"
## [1] "Computing chain 2"
```

```
preds.nngp = computePredictedValues(m.nngp,updater=list(GammaEta=FALSE))
MF.nngp = evaluateModelFit(hM=m.nngp, predY=preds.nngp)
MF.nngp
```

```
## $RMSE
## [1] 0.2892298 0.2116888 0.2745732 0.2611099 0.3012367
##
## $AUC
## [1] 0.9514327 0.9806548 0.9503898 0.9668750 0.9504831
##
## $TjurR2
## [1] 0.5129684 0.5592770 0.3735389 0.4825278 0.6192137
```

```
partition = createPartition(m.nngp, nfolds = 2, column = "sample")
cvpreds.nngp = computePredictedValues(m.nngp, partition=partition,updater=list(GammaEta=FALSE))
```

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

```
cvMF.nngp = evaluateModelFit(hM=m.nngp, predY=cvpreds.nngp)
cvMF.nngp
```

```
## $RMSE
## [1] 0.4067537 0.5056143 0.4538951 0.4730527 0.3073172
##
## $AUC
## [1] 0.7955318 0.6495536 0.6506024 0.6581250 0.9484702
##
## $TjurR2
## [1]  0.25037217 -0.13389507 -0.07299865 -0.06836150  0.58581198
```

Let's have a look at the MCMC trace plots for the spatial scale parameter `Alpha[[1]]`.

```
mpost.nngp = convertToCodaObject(m.nngp)
plot(mpost.nngp$Alpha[[1]])
```

```
summary(mpost.nngp$Alpha[[1]])
```

```
##
## Iterations = 1010:11000
## Thinning interval = 10
## Number of chains = 2
## Sample size per chain = 1000
```

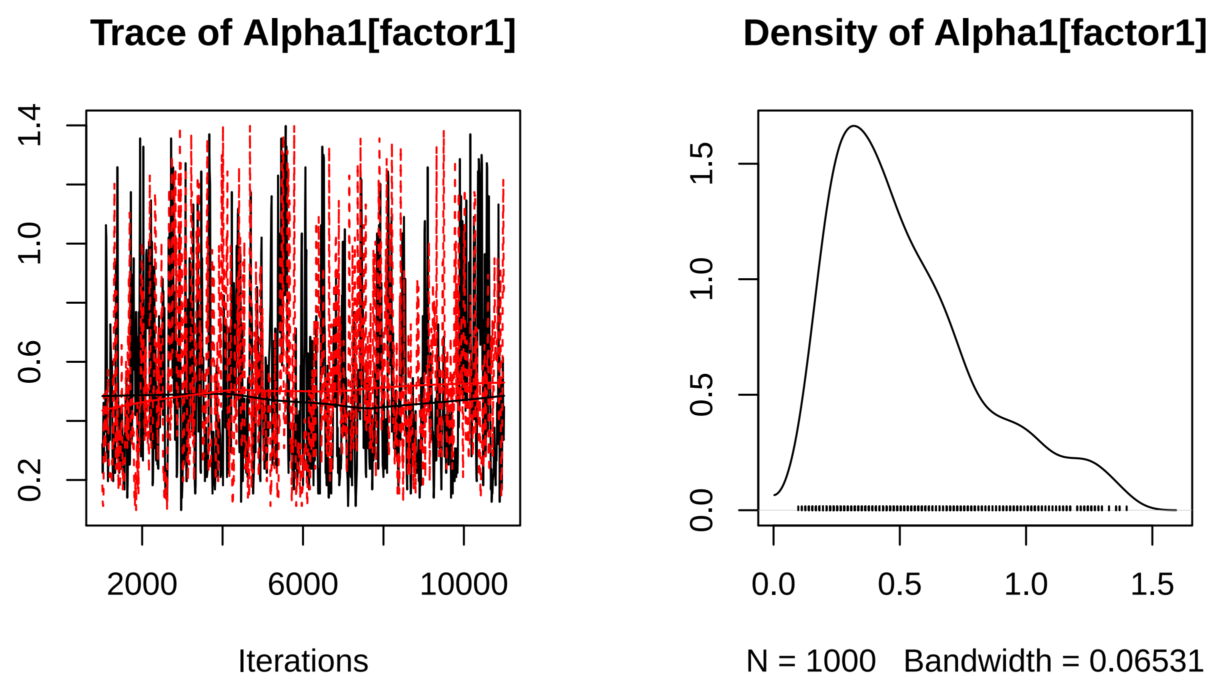

Figure 3: Posterior trace plot of the spatial scale parameter of the nngp spatial model.

```
##
## 1. Empirical mean and standard deviation for each variable,
##    plus standard error of the mean:
##
##           Mean             SD       Naive SE Time-series SE
##       0.527232       0.290593       0.006498       0.014648
##
## 2. Quantiles for each variable:
##
##   2.5%    25%    50%    75%  97.5%
## 0.1538 0.3076 0.4614 0.6852 1.2585
```

GPP models

The Gaussian Predictive Process (GPP) assumes that all information on the spatial structure of the data can be summarized at a small number of so called 'knot' locations. The locations of these knots have to be specified by the user. To help the user with this, we implemented a function to construct a uniform grid of knots based on the locations of the dataset. This function either need the wanted number of knots along the shortest spatial axis `nKnots` or the wanted distance between knots `KnotDist`. Additionally, the user can specify the maximum distance of a knot to the nearest data point, this ensures that the created grid does contain knots in locations with no datapoints.

```
# Setting the knots
Knots = constructKnots(xycoords, knotDist = 0.2, minKnotDist = 0.4)
```

```
plot(xycoords[,1],xycoords[,2],pch=18)
points(Knots[,1],Knots[,2],col='red',pch=18)
```

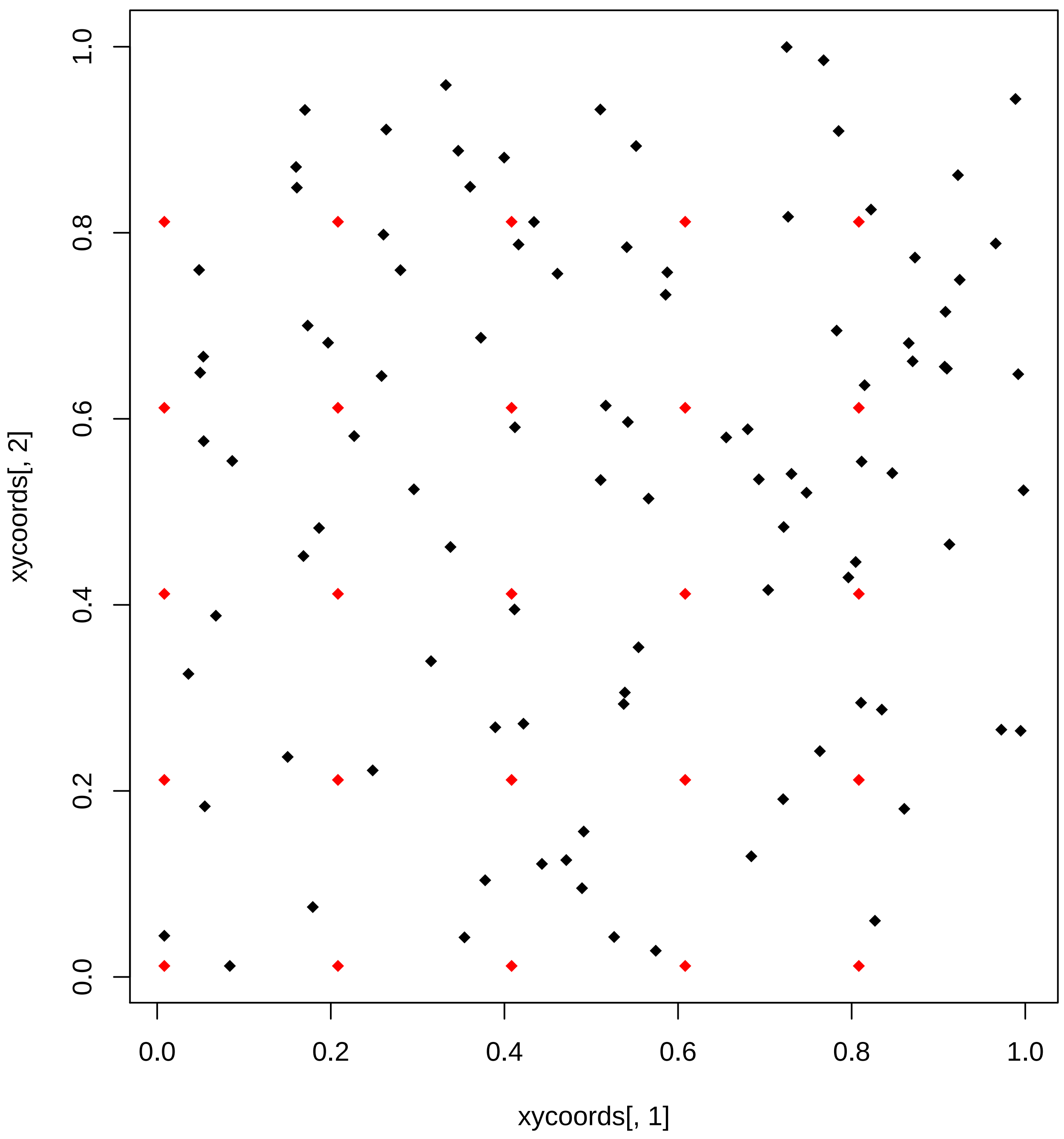

Figure 4: Locations of the created knots in red together with the locations of the plots in black.

We can now construct the random level by by setting `sMethod` to 'GPP' and supplying the knot locations to `sKnot`.

```r
rL.gpp = HmscRandomLevel(sData = xycoords, sMethod = 'GPP', sKnot = Knots)
rL.gpp = setPriors(rL.gpp,nfMin=1,nfMax=1)
m.gpp = Hmsc(Y=yprob, XData=XData, XFormula=~x1,
         studyDesign=studyDesign, ranLevels=list("sample"=rL.gpp),distr="probit")
m.gpp = sampleMcmc(m.gpp, thin = thin, samples = samples, transient = transient,
               nChains = nChains, verbose = verbose,updater=list(GammaEta=FALSE))
```

```
## [1] "Computing chain 1"
## [1] "Computing chain 2"
```

```r
preds.gpp = computePredictedValues(m.gpp,updater=list(GammaEta=FALSE))
MF.gpp = evaluateModelFit(hM=m.gpp, predY=preds.gpp)
MF.gpp
```

```
## $RMSE
## [1] 0.3028437 0.1820950 0.2570628 0.2595993 0.3010954
##
## $AUC
## [1] 0.9490044 0.9970238 0.9574770 0.9668750 0.9504831
##
## $TjurR2
## [1] 0.4769530 0.6052134 0.4170124 0.4651325 0.6195237
```

```r
cvpreds.gpp = computePredictedValues(m.gpp, partition=partition,updater=list(GammaEta=FALSE))
```

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

```r
cvMF.gpp = evaluateModelFit(hM=m.gpp, predY=cvpreds.gpp)
cvMF.gpp
```

```
## $RMSE
## [1] 0.3802496 0.3945502 0.3961586 0.4247680 0.3077173
##
## $AUC
## [1] 0.8470131 0.6465774 0.6413891 0.5793750 0.9472625
##
## $TjurR2
## [1]  0.26409357 -0.03237059 -0.02252235 -0.02332737  0.58537541
```

```r
mpost.gpp = convertToCodaObject(m.gpp)
plot(mpost.gpp$Alpha[[1]])
```

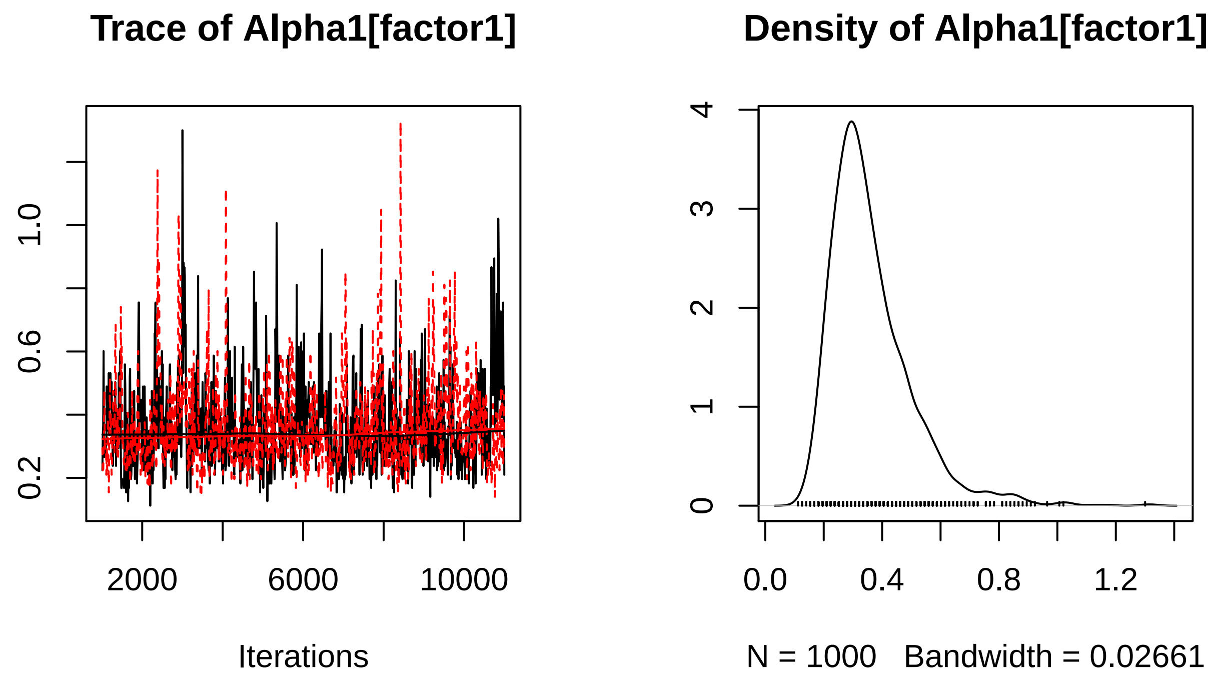

Figure 5: Posterior trace plot of the spatial scale parameter of the nngp spatial model.

```
summary(mpost.gpp$Alpha[[1]])
```

```
##
## Iterations = 1010:11000
## Thinning interval = 10
## Number of chains = 2
## Sample size per chain = 1000
##
## 1. Empirical mean and standard deviation for each variable,
##    plus standard error of the mean:
##
##           Mean             SD       Naive SE Time-series SE
##       0.361295       0.140674       0.003146       0.007020
##
## 2. Quantiles for each variable:
##
##   2.5%    25%    50%    75%  97.5%
## 0.1818 0.2657 0.3356 0.4195 0.7275
```

For this small dataset of 200 sampling units, you will notice that the sampling times for the three spatial models are very similar. However, for larger datasets, the sampling times of the NNGP and GPP method are much shorter than for the standard spatial model.

As a last remark, you may have noticed that we set the `updater` parameter in `sampleMCMC` when running the NNGP and GPP method. With this parameter we specify which MCMC updaters we want to include during the MCMC sampling. The standard setting is to use all available updaters. However, the GammaEta updater is currently not available for these methods. This is not a problem however because this is an optional updater which helps reach convergence in less iterations in some situations.

References

Tikhonov, G., L. Duan, N. Abrego, G. Newell, M. White, D. Dunson, and O. Ovaskainen. n.d. "Computationally Efficient Joint Species Distribution Modeling of Big Spatial Data."