Hmsc 3.0: Testing the performance of Hmsc with simulated data

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

10 August 2022

Introduction

The Hierarchical Modelling of Species Communities (Hmsc) framework 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 test the performance of the MCMC estimation scheme in terms of (1) computational time, (2) mixing properties, and (3) ability to recover true parameter values, by performing a set of experiments in which we use the Hmsc model to generate simulated data and then fit the model to the simulated data.

Generating simulated data

Set directories and load libraries

```
localDir = "."
source("load_libraries.R")
## we need to write results to subdirectories (folders)
if (!dir.exists(file.path(localDir, "data")))
  dir.create(file.path(localDir, "data"))
if (!dir.exists(file.path(localDir, "models")))
  dir.create(file.path(localDir, "models"))
if (!dir.exists(file.path(localDir, "performance")))
  dir.create(file.path(localDir, "performance"))
```

The `makedata` function

The `makedata` function included in the vignette folder produces datasets based on the Hmsc model. Options include

- ny = Number of sites
- ns = Number of species
- hierarchical = T/F, where T yields data with two nested random levels, and F yields a single random level.
- spatial = T/F, where T yields spatially explicit data with sample unit locations sampled from the unit square.

```
makedata = function(ns, ny, hierarchical=FALSE, spatial=FALSE)
```

Case 1: Baseline model

As the baseline case, we consider a community of n_s=50 species observed on n_y=200 sampling units. We include for each sampling unit two covariates, one categorical (each unit randomized to represent one of three classes with equal probability) and one continuous (sampled from standard normal distribution). We include for each species two traits, one categorical (each species randomized to represent one of three classes with equal probability) and one continuous (sampled from standard normal distribution). We assume that the species belong to groups of 5 species, and that the phylogenetic correlation is 0.9 within a group and 0 between the groups. We sampled each element of the matrix Γ (influence of traits on expected species parameters) from N(0, σ^2), set the matrix $\mathbf{V}$ (variation among species not explained by traits) to $(\sigma/2)^2$ $\mathbf{I}$, where we set σ=0.3 (for motivation behind this value, see below). We assumed the value ρ=0.5 for the phylogenetic signal parameter. We generated two sampling-level latent variables, both following standard normal distributions, and sampled the latent loadings on/for these variables from N(0, σ^2). We assumed that the data were normally distributed, obtained by adding to the linear predictor noise distributed as N(0,$(2\sigma)^2$). We set σ=0.3, resulting in the following empirical variances for vectorized quantities: variance of data 1.16, variance of fixed effects 0.60, variance of random effects 0.21, and residual variance 0.36. Out of the fixed effects, the expectation based on traits had variance 0.30. As the data are roughly equally influenced by all model components, we expected that true values of all model parameters can be recovered if a sufficient amount of data are available.

```
set.seed(1)
source("makedata.R")

ns=50
ny=200
tmp=makedata(ns=ns, ny=ny)
all.data=tmp[[1]]
all.parameters=tmp[[2]]
L1 = all.parameters$L
Y1 = all.data$Y

#First five species at first five sites
all.data$Y[1:5,1:5]
```

```
##         [,1]       [,2]       [,3]       [,4]         [,5]
## 1 -1.2925679 -0.2036966  0.2737067 -1.4590561 -1.223917388
## 2 -1.6421721  0.1790086  0.2027827 -2.5695424  0.504939800
## 3 -0.7262007 -0.5643915 -0.4290111 -1.4274946  0.015046529
## 4 -1.4849842 -0.1134727  1.8423574 -0.9821274 -0.006525039
## 5 -0.3883507  0.6396032 -1.5768310  0.8929502 -0.213049988
```

```
#Random levels for first five sites
all.data$study.design[1:5,]
```

```
##   sampling.unit     plot
## 1        su_001 plot_001
## 2        su_002 plot_002
## 3        su_003 plot_003
## 4        su_004 plot_004
## 5        su_005 plot_005
```

```
#Covariates
all.data$X.data[1:5,]
```

```
##   X.categorical X.covariate
## 1             A  0.44380533
## 2             A  0.05579711
## 3             C  0.06797239
## 4             B -0.20196174
## 5             C -1.15801525
```

```
#Trait data
all.data$Tr.data[1:5,]
```

```
##   Tr.categorical Tr.covariate
## 1              B   -0.1599017
## 2              A   -0.6257783
## 3              A    0.9004346
## 4              C   -0.9941936
## 5              A    0.8492504
```

```
c(var(as.vector(all.parameters$mu)), var(as.vector(all.parameters$LF)),
  var(as.vector(all.parameters$LR)), var(as.vector(all.parameters$eps)),
  var(as.vector(Y1)))
```

```
## [1] 0.3312244 0.3532705 0.2045964 0.3631548 0.9226622
```

```
save(file=file.path(localDir, "data","Case1.R"), all.data, all.parameters)
```

Case 2: Presence-absence model

We assumed the same linear predictor as for the baseline case but truncated the normally distributed data to 0/1 according to the probit model.

```
Y2 = 1*(L1 + matrix(rnorm(n = ny*ns), ncol=ns, nrow=ny)>0)
all.data$Y = Y2

#First five species at first five sites
all.data$Y[1:5,1:5]
```

```
##   [,1] [,2] [,3] [,4] [,5]
## 1    0    0    1    0    0
## 2    0    1    0    0    0
## 3    0    1    1    0    0
## 4    0    0    0    0    1
## 5    0    0    1    0    0
```

```
save(file=file.path(localDir, "data","Case2.R"), all.data, all.parameters)
```

Case 3: Log-normal Poisson model

We assumed the same model for the baseline case but converted the normally distributed data to follow the log-normal Poisson distribution. Thus, we sampled the data for Case 3 from the Poisson distribution, the expectation of which was set to the exponential of the data from Case 1.

```
Y3 <- rpois(length(Y1), exp(Y1))
dim(Y3) <- dim(Y1)
all.data$Y = Y3

#First five species at first five sites
all.data$Y[1:5,1:5]
```

```
##      [,1] [,2] [,3] [,4] [,5]
## [1,]    0    2    2    0    2
## [2,]    0    2    1    0    3
## [3,]    1    4    0    0    1
## [4,]    0    1    5    0    0
## [5,]    1    2    0    1    1
```

```
save(file=file.path(localDir, "data","Case3.R"), all.data, all.parameters)
```

Case 4: Large number of sites

We assumed otherwise the same model as for Case 1 but set the number of sampling units to n_y=2000.

```
tmp=makedata(ns=50, ny=2000)
all.data=tmp[[1]]
all.parameters=tmp[[2]]
save(file=file.path(localDir, "data","Case4.R"), all.data, all.parameters)
```

Case 5: Large number of species

We assumed otherwise the same model as for Case 1 but set the number of species to n_s=200.

```
tmp=makedata(ns=200, ny=200)
all.data=tmp[[1]]
all.parameters=tmp[[2]]
save(file=file.path(localDir, "data","Case5.R"), all.data, all.parameters)
```

Case 6: Spatially structured data

We assumed otherwise the same model as for Case 1 but included the coordinates of the sampling units, which were sampled uniformly from the unit square. The latent variables were assumed to have exponentially decaying spatial correlation structure, with the scale of decay set to 0.5 spatial units for latent variable 1 and 0.1 spatial units for latent variable 2.

```
tmp=makedata(ns=50, ny=200, spatial=TRUE)
all.data=tmp[[1]]
all.parameters=tmp[[2]]

#Coordinates for first five sites
all.data$xy[1:5,]
```

```
##             [,1]      [,2]
## [1,] 0.271291679 0.1825577
## [2,] 0.332560229 0.4316660
## [3,] 0.510723566 0.8143120
## [4,] 0.021381523 0.7597966
## [5,] 0.001947572 0.9109282
```

```
save(file=file.path(localDir, "data","Case6.R"), all.data, all.parameters)
```

Case 7: Hierarchically structured data

We assumed otherwise the same model as for Case 1 but assumed that the sampling units belonged to plots, each plot containing 10 sampling units. We assumed one latent variable at the plot level and another at the sampling unit level, thus keeping the variance attributed to latent variables the same as in Case 1 but changing their structure.

```
tmp=makedata(ns=50, ny=200, hierarchical=TRUE)
all.data=tmp[[1]]
all.parameters=tmp[[2]]

#Hierarchical random levels
head(all.data$study.design[order(all.data$study.design$plot),][c(1:3,21:23),])
```

```
##    sampling.unit     plot
## 1         su_001 plot_001
## 11        su_011 plot_001
## 21        su_021 plot_001
## 2         su_002 plot_002
## 12        su_012 plot_002
## 22        su_022 plot_002
```

```
save(file=file.path(localDir, "data","Case7.R"), all.data, all.parameters)
```

Model setup and fitting

We first set the parameters that we used in posterior samplings. There is an alternative for `test.run` that runs fast, but gives no real results, but you can use to quickly see the idea.

```
nChains = 4
test.run = FALSE
if (test.run){
   #with this option, the vignette evaluates in ca. ? minutes in a laptop
   thin = 1
   samples = 10
} else {
   #with this option, the vignette evaluates in ca. ? hrs in a laptop
   thin = 10
   samples = 1000
}
verbose = 0
```

We fitted `Hmsc` models with structure matching the data generation model, i.e. we did not examine here the influence of model misspecification or e.g. issues related to variable selection. However, we did not fix the number of latent variables to that used for generating the data, but assumed the default prior for them, as well as for all other model parameters.

We fitted each model with sampling parameters set to thin=10 and samples=1000, and thus we performed 1.5×10^4 MCMC iterations per chain. We adapted the number of latent factors during the first 4000 iterations, and ignored from each chain the first 5000 iterations as transient. We used 4 chains, so we ran in total 6×10^4 MCMC iterations, out of which the thinned posterior sample consisted of 4000 samples.

The code below takes a long time to run. The .Rmd file can be run much more quickly by setting `test.run=TRUE`, but the results will not be reliable.

For case 1, 4 and 5, we set up the model with gaussian errors, using `distr = "normal"`

```
for (case in c(1,4,5)){
  set.seed(1)
  load(file = file.path(localDir, "data", paste("Case",toString(case),".R", sep="")))
  m = Hmsc(Y = all.data$Y,
           XData = all.data$X.data, XFormula = all.data$X.formula,
           TrData = all.data$Tr.data, TrFormula = all.data$Tr.formula,
           C = all.data$C,
           distr = "normal", studyDesign = all.data$study.design,
           ranLevels = list(
             "sampling.unit" = HmscRandomLevel(units = all.data$study.design[,1])))
  ptm = proc.time()
  m = sampleMcmc(m, samples=samples, thin=thin, transient = ceiling(0.5*samples*thin),
                 nChains = nChains, nParallel = nChains, verbose=verbose)
  computational.time =  proc.time() - ptm
  save(file=file.path(localDir, "models", paste("Case",toString(case),".R", sep="")),
       m, computational.time)
}
```

For case 2, we set up the model with probit errors, using `distr = "probit"`

```
set.seed(1)
load(file=file.path(localDir, "data","Case2.R"))
m = Hmsc(Y=all.data$Y,
         XData=all.data$X.data, XFormula=all.data$X.formula,
         TrData=all.data$Tr.data, TrFormula=all.data$Tr.formula,
         C=all.data$C,
         distr="probit", studyDesign=all.data$study.design,
         ranLevels=list(
           "sampling.unit"=HmscRandomLevel(units = all.data$study.design[,1])))
ptm = proc.time()
m = sampleMcmc(m, samples=samples, thin=thin, transient = ceiling(0.5*samples*thin),
                 nChains = nChains, nParallel = nChains, verbose=verbose)
computational.time =  proc.time() - ptm
save(file=file.path(localDir, "models","Case2.R"), m, computational.time)
```

For case 3, we set up the model with lognormal Poisson errors, using `distr = "lognormal poisson"`

```
set.seed(1)
load(file=file.path(localDir, "data","Case3.R"))
m = Hmsc(Y=all.data$Y,
```

```
         XData=all.data$X.data, XFormula=all.data$X.formula,
         TrData=all.data$Tr.data, TrFormula=all.data$Tr.formula,
         C=all.data$C,
         distr="lognormal poisson", studyDesign=all.data$study.design,
         ranLevels=list(
           "sampling.unit"=HmscRandomLevel(units = all.data$study.design[,1])))
ptm = proc.time()
m = sampleMcmc(m, samples=samples, thin=thin, transient = ceiling(0.5*samples*thin),
               nChains = nChains, nParallel = nChains, verbose=verbose)
computational.time =  proc.time() - ptm
save(file=file.path(localDir, "models","Case3.R"), m, computational.time)
```

For case 6, we set up the model with a spatially structured random level, using `HmscRandomLevel(data=sRL, priors="default")`. Note that `sData` is used instead of `units`.

```
set.seed(1)
load(file=file.path(localDir, "data","Case6.R"))
sRL = all.data$xy
colnames(sRL) = c("x","y")
rownames(sRL) = all.data$study.design[,1]
rL = HmscRandomLevel(sData=sRL)

m = Hmsc(Y=all.data$Y,
         XData=all.data$X.data, XFormula=all.data$X.formula,
         TrData=all.data$Tr.data, TrFormula=all.data$Tr.formula,
         C=all.data$C,
         distr="normal", studyDesign=all.data$study.design,
         ranLevels=list("sampling.unit"=rL))
ptm = proc.time()
m = sampleMcmc(m, samples=samples, thin=thin, transient = ceiling(0.5*samples*thin),
               nChains = nChains, nParallel = nChains, verbose=verbose)
computational.time =  proc.time() - ptm
save(file=file.path(localDir, "models","Case6.R"), m, computational.time)
```

For case 7, we set up two hierarchical random levels, using `HmscRandomLevel(units=...)`.

```
set.seed(1)
load(file=file.path(localDir, "data","Case7.R"))
m = Hmsc(Y=all.data$Y,
         XData=all.data$X.data, XFormula=all.data$X.formula,
         TrData=all.data$Tr.data, TrFormula=all.data$Tr.formula,
         C=all.data$C,
         distr="normal", studyDesign=all.data$study.design,
         ranLevels=list(
           "sampling.unit"=HmscRandomLevel(units = all.data$study.design[,1]),
           "plot"=HmscRandomLevel(units = all.data$study.design[,2]))
         )
ptm = proc.time()
m = sampleMcmc(m, samples=samples, thin=thin, transient = ceiling(0.5*samples*thin),
               nChains = nChains, nParallel = nChains, verbose=verbose)
computational.time =  proc.time() - ptm
save(file=file.path(localDir, "models","Case7.R"), m, computational.time)
```

Evaluating model performance

MCMC mixing

We used the `coda` package to compute the effective number of MCMC samples (sample size adjusted for autocorrelation) and Potential Scale Reduction Factor (Gelman and Rubin's convergence diagnostic, PSRF) for the following parameters: β, γ, ρ, V, Ω. Note the following useful functions

- `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).
- `convertToCodaObject` converts the posterior distributions produced by `Hmsc` into `coda`-format.

```
ptm = proc.time()

cases = c(1,2,3,4,5,6,7)

for (case in cases){
  print(case)
  set.seed(1)
  load(file=file.path(localDir, "data", paste("Case",toString(case),".R", sep="")))
  load(file=file.path(localDir, "models", paste("Case",toString(case),".R", sep="")))
  computational.time = computational.time[3]

  predY = computePredictedValues(m, expected=FALSE)
  MF = evaluateModelFit(m, predY)

  # ASSESS MIXING
  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

  mpost$temp = mpost$Omega[[1]]
  for(i in 1:length(mpost$temp)){
    mpost$temp[[i]] = mpost$temp[[i]][,1:50^2]
  }

  es.omega = effectiveSize(mpost$temp)
  ge.omega = gelman.diag(mpost$temp,multivariate=FALSE)$psrf
  es.alpha = NA
  ge.alpha = NA
```

```
  if (case==6){
    es.alpha = effectiveSize(mpost$Alpha[[1]])
    ge.alpha = gelman.diag(mpost$Alpha[[1]])
  }

  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)

# COMPUTE NRMSE AND COR
  estimates = list()
  postList=poolMcmcChains(m$postList, start = 1)
  npost = length(postList)
  npri = npost

  for (para in 1:5){
    parameter = c("beta","gamma","rho","V","omega")[para]
    if (parameter=="beta"){
      getpar = function(a)
        return(a$Beta)
      true = all.parameters$beta
      prior = list()
      for(i in 1:npost){
        gamma = matrix(MASS::mvrnorm(n=1, mu=m$mGamma, Sigma = m$UGamma), ncol=m$nt, nrow=m$nc)
        mu = tcrossprod(gamma,m$Tr)
        rho = sample(size=1,x=m$rhopw[,1],prob=m$rhopw[,2])
        V = solve(MCMCpack::rwish(m$f0, solve(m$V0)))
        Si = kronecker(V,rho*all.data$C + (1-rho)*diag(m$ns))
        prior[[i]] = matrix(MASS::mvrnorm(n=1, mu=as.vector(mu), Sigma=Si), ncol=m$ns, nrow=m$nc)
      }
    }

    if (parameter=="rho"){
      getpar = function(a)
        return(a$rho)
      true = all.parameters$rho
      prior = list()
      for(i in 1:npri){
        prior[[i]] = sample(size=1,x=m$rhopw[,1],prob=m$rhopw[,2])
      }
    }

    if (parameter=="gamma"){
      getpar = function(a)
        return(a$Gamma)
      true = all.parameters$gamma
      prior = list()
      for(i in 1:npri){
        prior[[i]] = matrix(MASS::mvrnorm(n=1, mu=m$mGamma, Sigma = m$UGamma), ncol=m$nt, nrow=m$nc)
      }
    }
```

```
if (parameter=="V"){
  getpar = function(a)
    return(a$V)
  true = all.parameters$V
  prior = list()
  for(i in 1:npost){
    prior[[i]] = solve(MCMCpack::rwish(m$f0, solve(m$V0)))
  }
}

if (parameter=="omega"){
  getpar = function(a)
    return(t(a$Lambda[[1]])%*%a$Lambda[[1]])
  true = t(all.parameters$lambda)%*%all.parameters$lambda
  nu = m$rL[[1]]$nu
  a1 = m$rL[[1]]$a1
  a2 = m$rL[[1]]$a2
  b1 = m$rL[[1]]$b1
  b2 = m$rL[[1]]$b2
  prior = list()
  for(i in 1:npost){
    nf = 10
    delta = rep(NA,nf)
    delta[1] = rgamma(1,a1,b1)
    for (j in 2:nf){
      delta[j] = rgamma(1,a2,b2)
    }
    tau = cumprod(delta)
    psi = matrix(rgamma(nf*m$ns, nu/2, nu/2), nf, m$ns)
    sd = sqrt(1/(psi*matrix(rep(tau,m$ns),nr=nf, nc=m$ns)))
    la = rnorm(length(sd), mean=0, sd=sd)
    dim(la) = dim(sd)
    prior[[i]] = t(la)%*%la
  }
}

post = lapply(postList, getpar)
post.mean = Reduce("+", post) / length(post)
cors = rep(0,npost)
for (i in 1:npost){
  cors[i] = cor(as.vector(true),as.vector(post[[i]]))
}
se = function(a)
  return((a-true)^2)
post.se = lapply(post, se)
prior.se = lapply(prior, se)
post.rmse = sqrt(Reduce("+", post.se) / length(post.se))
prior.rmse = sqrt(Reduce("+", prior.se) / length(prior.se))
nrmse = post.rmse / prior.rmse

if (parameter=="beta"){
  estimates$cor.beta = cors
  estimates$nrmse.beta = nrmse
```

```
    }
    if (parameter=="gamma"){
      estimates$cor.gamma = cors
      estimates$nrmse.gamma = nrmse
    }
    if (parameter=="rho"){
      estimates$post.rho = post
      estimates$nrmse.rho = nrmse
    }
    if (parameter=="V"){
      estimates$cor.V = cors
      estimates$nrmse.V = nrmse
    }
    if (parameter=="omega"){
      estimates$cor.omega = cors
      estimates$nrmse.omega = nrmse
    }
  }

  save(file=file.path(localDir, "performance",paste("Case",toString(case),".R",sep="")),
       computational.time, mixing, estimates, MF)
}
```

Computational time

The timings and relative timings vary a lot. They are strongly dependent on the hardware and its load, and our example cannot be generalized to other environments. Parallel processing can speed up calculations, but we can only run each chain in parallel, and the slowest chain will determine the running time. Most of time is spent in linear algebra in external libraries BLAS and LAPACK. BLAS provides the basic matrix operations and LAPACK the higher level matrix algorithms. R ships with so-called reference libraries for both, and for BLAS it is called libRblas. Changing the reference libraries to implementation optimized for the hardware can speed up calculations by a factor of ten. OpenBLAS and associated LAPACK is a very popular choice, but there are many other alternatives (see R manual on Installation and Administration and R FAQs). Optimized BLAS and LAPACK libraries may use parallel processing and they can use several cores for each parallel chain. The number of available cores depends also on the system configuration, and especially in multi-user systems, on other CPU load and system policies. We used unknown BLAS and ran 4 chains in parallel in a system with 8 cores.

Table 1: Computational times to perform the MCMC sampling in minutes and relative to the time of the baseline case

	1	2	3	4	5	6	7
Time (min)	6.07	4.80	5.74	8.42	89.86	69.38	11.00
Time (relative)	1.00	0.79	0.95	1.39	14.81	11.43	1.81

The effective sample size (Figure 1) was close to 4000 for the β, γ, and V parameters, except for Case 3 (the lognormal Poisson model) and to some extent Case 5 (large number of species), for which cases a longer MCMC chain would be needed for obtaining robust results. Mixing was more challenging to obtain for the Ω and ρ parameters, for which the effective sample size varied much among the elements of the matrix also for the normally distributed models.

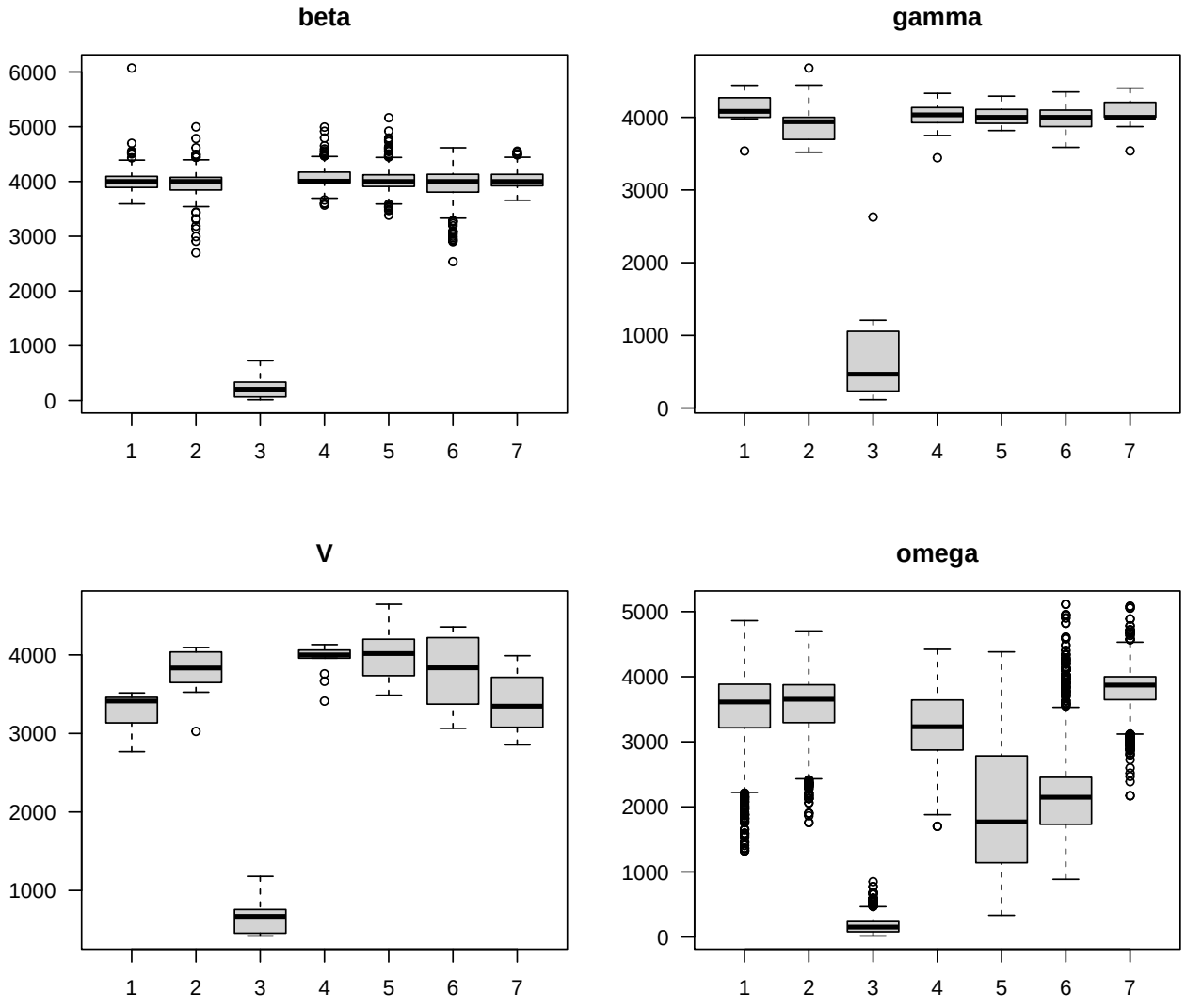

Figure 1: Distributions of effective sample sizes of multivariate parameters.

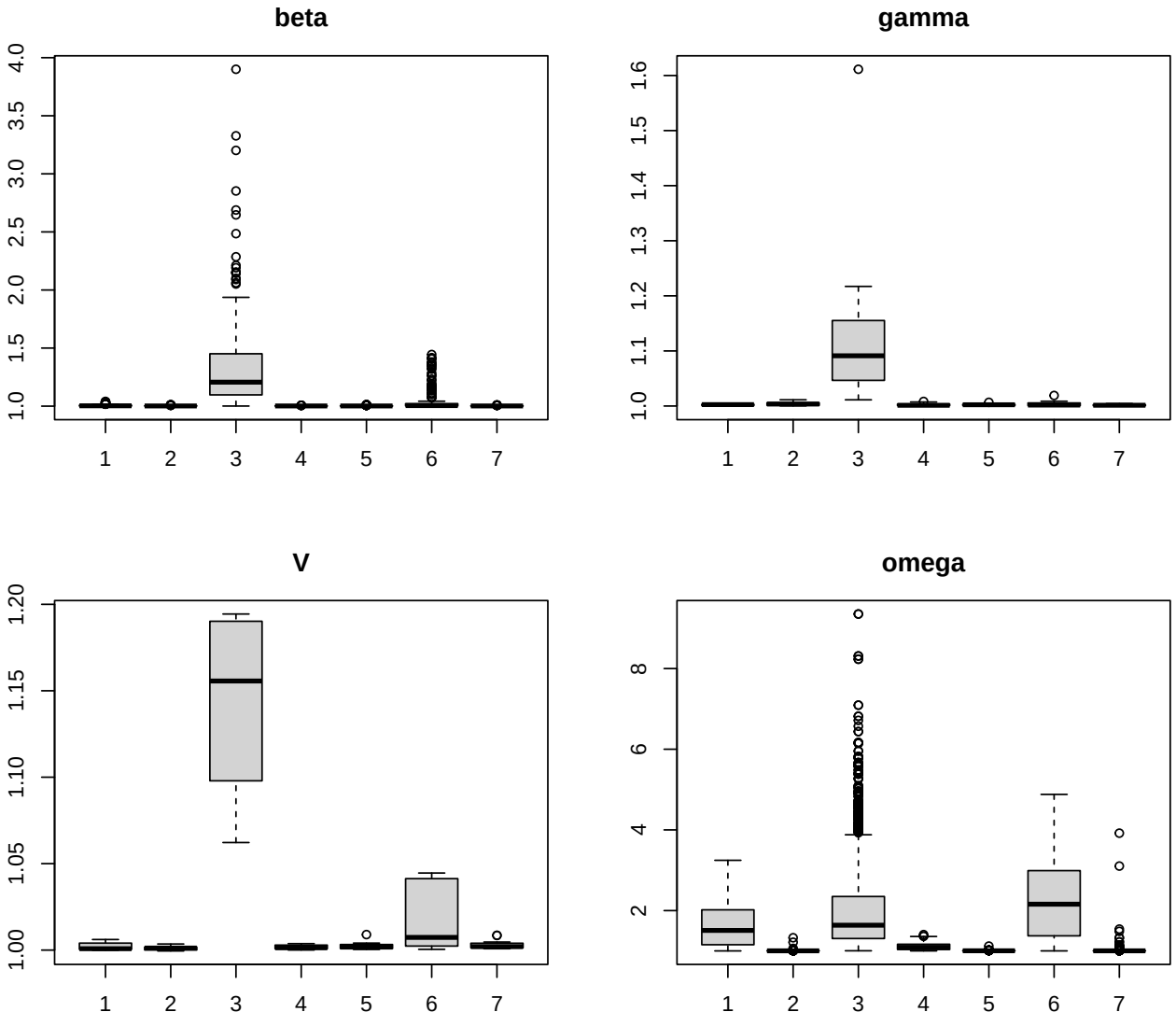

Figure 2: Distributions of upper C.I. for potential scale reduction factors for multivariate parameters.

Ability to recover true parameter values

For the same parameters that we considered for MCMC mixing, we asked how much closer the posterior distributions were to the true values than were the prior distributions. We quantified this by the normalized root mean squared errors (NRMSE), where the normalization was done with respect to the root mean squared error under the prior distribution. Further, for the matrix-valued parameters β, γ, ρ, V, Ω we computed the posterior distribution of the Mantel correlation between the estimated and true values. For the univariate parameter ρ we compared the true value to the boxplot of the posterior distribution.

The species-specific parameters β were better estimated for normal models than for probit and lognormal Poisson models (Figure 3). As expected, they were most accurately estimated for Case 4 with many sampling units. For the community-level parameters γ the difference among the cases was smaller, the probit model performing slightly worse than the others. The Ω parameters were estimated best for normal models where the latent variables were independent among the sampling units. The V parameters were roughly equally well estimated in all cases. In all cases the estimation of the phylogenetic signal parameter ρ was difficult, in particular for the probit model were the posterior distribution was essentially identical to the prior (Figure 5).

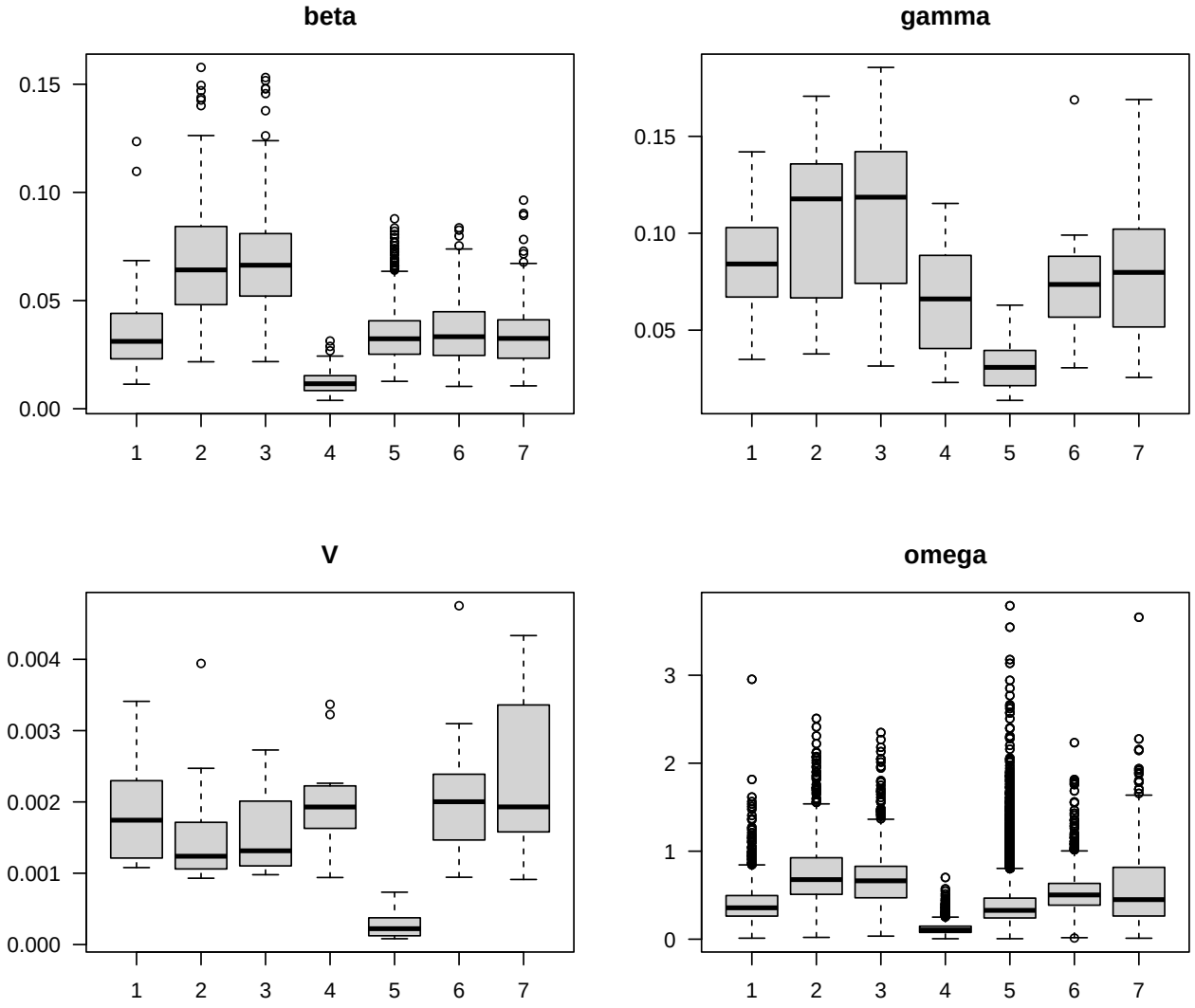

Figure 3: Distributions of normalized root mean squared errors (NRMSE) for multivariate parameters. Values smaller than one indicate that the posterior distribution is closer to the true value than the prior distribution.

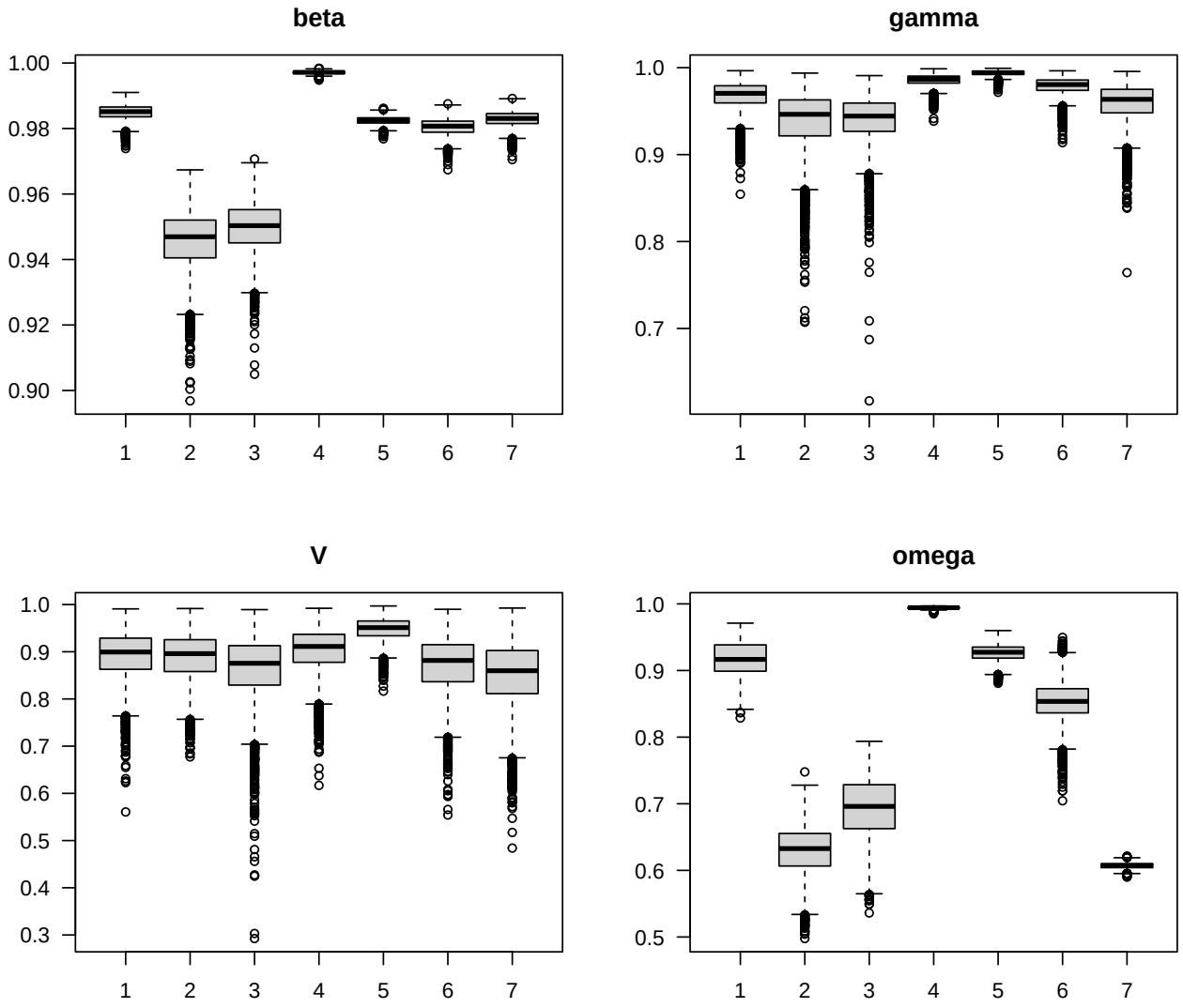

Figure 4: Posterior distributions of correlation between true and estimated values for multivariate parameters.

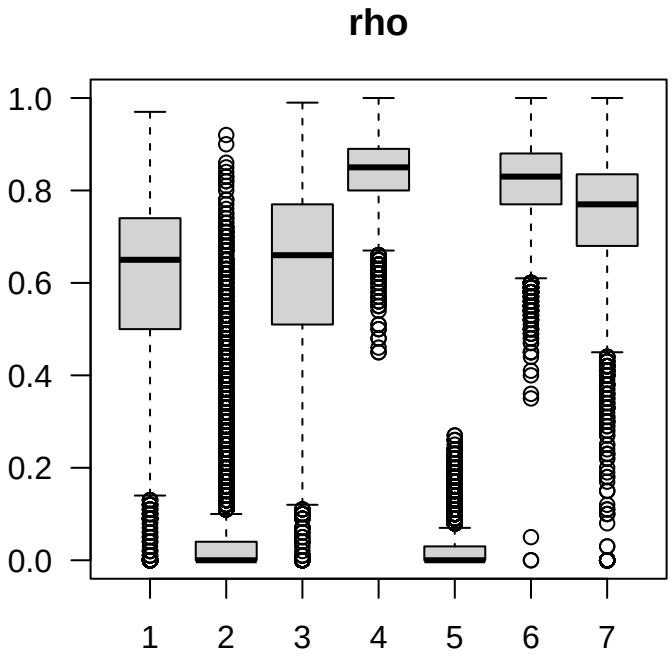

Figure 5: Posterior distributions of the phylogenetic signal parameter.

Table 2: Performance of HMSC-R for estimating the phylogenetic signal parameter.

	1	2	3	4	5	6	7
Effective sample size	2181.00	2009.00	149.00	3415.00	4793.00	2674.00	2433.00
Upper C.I. of PSRF	1.00	1.00	1.17	1.00	1.00	1.02	1.00
NRMSE	0.59	1.11	0.59	0.85	1.17	0.81	0.69

References

Ovaskainen, O., A. Tikhonov, A. Norberg, F.G. Blanchet, L. Duan, D. Dunson, T. Roslin, and N. Abrego. 2017. "Computationally Efficient Joint Species Distribution Modeling of Big Spatial Data." *Ecology Letters* 20: 561–76. doi:10.1111/ele.12757.