Processing and Analysis of Biological Data

Bayesian methods

Øystein H. Opedal

10 Dec 2023

Bayesian methods

All the modelling approaches we have discussed so far have involved models fitted by least-square or maximum likelihood methods. This is not the only method for fitting models to data though. In this section we will discuss the philosophy of Bayesian statistics, and some examples of how models can be fitted using Bayesian inference.

Put simply, while hypothesis testing in "frequentist statistics" is based on estimating the most likely value of parameters, their uncertainty, and possibly P-values, the Bayesian philosophy is explicitly to consider the distribution of plausible parameter values. Furthermore, Bayesian inference involves the formulation of a so-called *prior distribution* that describes our prior belief about which parameter values are more likely to occur.

At the core of Bayesian inference sits the simple Bayes theorem or Bayes rule, which states

$P(A|B) = \frac{P(B|A)P(A)}{P(B)}$

Here A and B are events, and the $P(A)$ and $P(B)$ represent the prior belief about the events in the absence of additional information.

In the context of model fitting, we replace the events A and B with the model parameters θ and the observed data S.

$P(\theta|S) = \frac{P(S|\theta)P(\theta)}{\int P(S|\theta)P(\theta)d\theta}$

where $P(\theta)$ is the prior for the parameters and $P(S|\theta)$ is the likelihood function.

Bayesian model fitting occurs in an iterative process, normally the Monte Carlo Markov Chain (MCMC). A simple method for model fitting by MCMC is Metropolis-Hastings updating. Briefly, during each iteration a small modification is done to the current value of a parameter, and if this value increases the likelihood of the model (after taking into account the prior for the parameter), the suggestion is accepted and the chain makes the next suggestion based on this updated value. If the suggestion does not improve the likelihood, it is discarded. Through this iterative process, the chain eventually reaches a stable distribution of plausible parameter values, the *posterior distribution.* For complex models this method is generally not feasible, but more sophisticated updating procedures are used that essentially performs the same task (e.g. Gibbs sampling).

The parameter estimates and their uncertainty can be summarized for example as the posterior mean and a credible interval around the mean. Credible intervals in Bayesian analysis corresponds to confidence intervals in maximum-likelihood analyses, but are termed differently due to the difference in how they are obtained.

Instead of computing summaries based on the posterior distribution, we can perform downstream analyses for each posterior sample, thus carrying the uncertainty forward and subsequently obtain e.g. posterior means and credible intervals after all analysis steps are complete.

Though Bayesian inference differs philosophically and technically from maximum-likelihood analysis, many applications are ultimately rather similar. For example, we can use Bayesian inference to fit linear models using (nearly) non-informative priors, and obtain nearly identical parameter estimates to those obtained by the `lm` function. Statistical support can be evaluated as e.g. the posterior support, i.e. the proportion of posterior samples that are greater than zero.

Some 'purists' consider themselves strictly 'frequentists' or strictly 'Bayesians', but most (the author included) are more than happy to leverage the strengths of specific methods in specific situations.

Fitting a linear model with Bayesian inference

To demonstrate the Bayesian model-fitting procedure, we will use the `Hmsc` package. `Hmsc` stands for Hierarchical Modelling of Species Communities, and implements a modelling framework developed primarily for analyses of multivariate community data. In a later section we will explore multivariate versions of `Hmsc`, but here we will use the package to fit a more standard mixed model. Another option for fitting such models is the `MCMCglmm` package.

The following examples are similar to those given in a vignette associated with the `Hmsc`package, which can be obtained by calling `vignette("vignette_1_univariate", package="Hmsc)`.

As a first very simple example, we will fit a simple linear regression to simulated data. While the `lm` function constructs and fits the model in one go, the `Hmsc`package first constructs the model, and then performs model fitting (posterior sampling) in a second step. This is because posterior sampling can take a long time for complex data, and we want to be able to leave it to run e.g. overnight. For the current model though, model fitting is very quick.

```
library(Hmsc)

x = rnorm(200, 10, 3)
y = -2 + 0.4*x + rnorm(200, 0, 2)

m1 = lm(y~x)
m2 = Hmsc(Y = as.matrix(y), XData = data.frame(x), XFormula = ~x,
        distr="normal")

m2 = sampleMcmc(m2, samples=1000, transient=1000, thin=1,
                nChains=2, verbose=F)
```

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

```
summary(m1)$coef
```

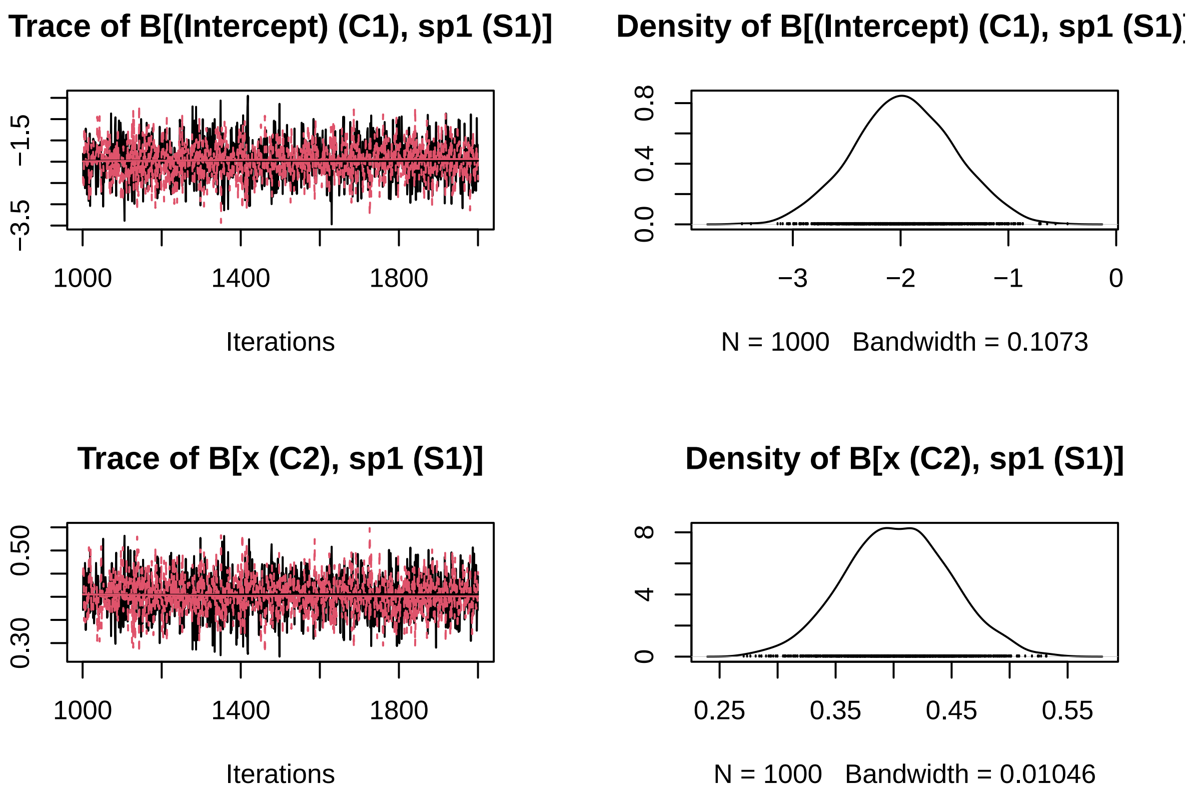
```
##              Estimate Std. Error   t value     Pr(>|t|)
## (Intercept) -1.990347 0.47522100 -4.188256 4.229733e-05
## x            0.407529 0.04695017  8.680032 1.455903e-15
```

```
mpost = convertToCodaObject(m2)
summary(mpost$Beta)
```

```
##
## Iterations = 1001:2000
## Thinning interval = 1
## 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
## B[(Intercept) (C1), sp1 (S1)] -1.9669 0.46300 0.010353       0.010826
## B[x (C2), sp1 (S1)]            0.4035 0.04512 0.001009       0.001042
##
## 2. Quantiles for each variable:
##
##                                   2.5%     25%     50%    75%   97.5%
## B[(Intercept) (C1), sp1 (S1)] -2.8820 -2.2783 -1.9704 -1.648 -1.0578
## B[x (C2), sp1 (S1)]            0.3157  0.3721  0.4028  0.434  0.4948
```

The parameter estimates are very similar, though not identical. This is due to the stochasticity of the MCMC algorithm. The fact that we have sampled the posterior distribution with MCMC also means that, before we start looking more in detail at the model estimates, we should assess whether the model actually converged on a stable solution. One way to do this is to produce a posterior trace plot, which shows how the parameter estimates have changed during the MCMC chain.

```
plot(mpost$Beta)
```



Here the posterior trace plot looks fine. There is no directional trend and the posterior distribution is nicely bell-shaped. We can also evaluate whether the chain has mixed (explored the parameter space) well by computing the effective sample size (which should be close to the number of posterior samples).

```r
effectiveSize(mpost$Beta)
```

```
## B[(Intercept) (C1), sp1 (S1)]           B[x (C2), sp1 (S1)] 
##                      1848.791                      1885.987
```

Because we ran two independent MCMC chains, we can also assess whether they yielded similar results, which can be quantified by the so-called *potential scale reduction factor* or the Gelman-Rubin criterion. Values close to 1 means that the chains yielded similar results.

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

```
##                               Point est. Upper C.I.
## B[(Intercept) (C1), sp1 (S1)]   1.000609   1.001108
## B[x (C2), sp1 (S1)]             1.000568   1.002335
```

One advantage of Bayesian analyses is that, because we obtain the entire posterior distribution, we can carry uncertainty in parameter estimates forward to subsequent steps in the analysis. As a simple example, if we are using the parameter estimates of the simple linear model above to make predictions about the value of y for some values of x, we can obtain those predictions across the entire posterior distribution, and thus construct e.g. a confidence interval for the predictions. Because 'confidence intervals' are well defined in standard maximum-likelihood statistics, Bayesians refer instead to 'credible intervals', often so-called 'highest posterior density' (HPD) intervals.

Fitting a linear mixed model

As a second example, we will fit a mixed model with random intercepts. Again, we will also fit the same model using maximum likelihood.

```r
library(glmmTMB)

set.seed(145)
x1 = rnorm(200, 10, 2)

groupmeans = rep(rnorm(10, 20, 4), each=20)
groupID = as.factor(rep(paste0("Group", 1:10), each=20))

y = 2 + 1.5*x1 + groupmeans + rnorm(200, 0, 2)

m1 = glmmTMB(y~x1 + (1|groupID))
```

```
## Warning in glmmTMB(y ~ x1 + (1 | groupID)): use of the 'data' argument is
## recommended
```

To define a mixed model with the `Hmsc` package, we first create a data frame containing the group IDs (`studyDesign`), and then define a random effect using the `HmscRandomLevel` function. As above, we first define the model and then sample the posterior distribution.

```
studyDesign = data.frame(group = as.factor(groupID))
rL1 = HmscRandomLevel(units = groupID)

m2 = Hmsc(Y = as.matrix(y), XData = data.frame(x1), XFormula = ~x1,
          studyDesign = studyDesign, ranLevels = list(group = rL1),
          distr="normal")

m2 = sampleMcmc(m2, samples=1000, transient=1000, thin=1,
                nChains=2, nParallel=2, verbose=F)
```

For the glmmTMB model, we get both the parameter estimates and random-effect variances from the summary function. For the Hmsc model, we get the parameter estimates and variance explained by the random effect (group) separately (the latter is called Omega in Hmsc).

```
summary(m1)
```

```
##  Family: gaussian  ( identity )
## Formula:          y ~ x1 + (1 | groupID)
##
##      AIC      BIC   logLik deviance df.resid
##    896.5    909.7   -444.2    888.5      196
##
## Random effects:
##
## Conditional model:
##  Groups   Name        Variance Std.Dev.
##  groupID  (Intercept) 10.576   3.252
##  Residual              4.081   2.020
## Number of obs: 200, groups:  groupID, 10
##
## Dispersion estimate for gaussian family (sigma^2): 4.08
##
## Conditional model:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept)  23.8343     1.3018   18.31   <2e-16 ***
## x1            1.4528     0.0764   19.02   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
mpost = convertToCodaObject(m2)
summary(mpost$Beta)
```

```
##
## Iterations = 1001:2000
## Thinning interval = 1
## 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
## B[(Intercept) (C1), sp1 (S1)] 23.765 1.30056 0.029081       0.033277
## B[x1 (C2), sp1 (S1)]           1.449 0.07525 0.001683       0.001683
##
## 2. Quantiles for each variable:
##
##                                  2.5%    25%    50%    75%  97.5%
## B[(Intercept) (C1), sp1 (S1)] 21.293 22.831 23.730 24.636 26.287
## B[x1 (C2), sp1 (S1)]           1.303  1.399  1.449  1.498  1.598
```

```
getPostEstimate(m2, "Omega")$mean
```

```
##          [,1]
## [1,] 11.02465
```

As in the previous example, all the estimated parameters are very similar.

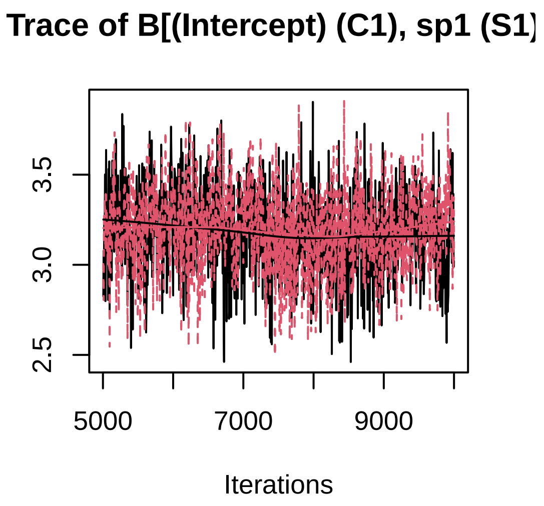
Exercise

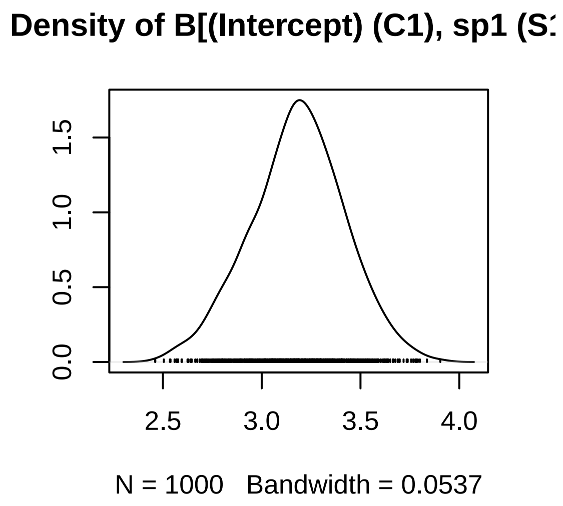
Revisit the *Eulaema* dataset from the GLM exercise. Fit a mixed model with the `Hmsc` package, including study area as a random effect. Negative binomial errors are not available in `Hmsc`, but Poisson errors are (`distr="poisson"`). You can also consider the "lognormal poisson" distribution, which is also tailored to zero-inflated data. Remember to assess model convergence before looking at the parameter estimates. If the model has not converged, increase the thinning interval between each posterior sample (but gradually, you don´t want to spend hours fitting the model).

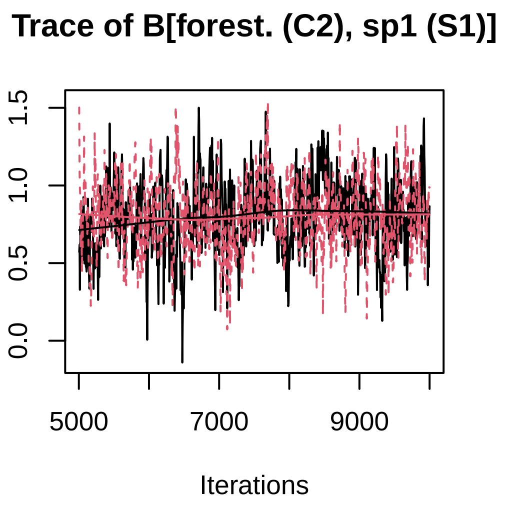
```
dat = read.csv("datasets/Eulaema.csv")

dat$SA = as.factor(dat$SA)
dat$method = as.factor(dat$method)
XData = data.frame(dat[,4:12])

studyDesign = data.frame(SA = dat$SA)
rL1 = HmscRandomLevel(units = dat$SA)
```

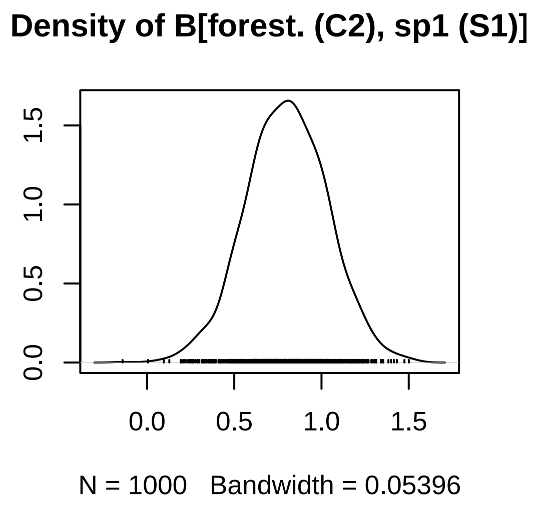
```
m2 = Hmsc(Y = as.matrix(dat$Eulaema_nigrita), XData = XData, XFormula = ~forest.,
          studyDesign = studyDesign, ranLevels = list(SA = rL1),
          distr="poisson")

m2 = sampleMcmc(m2, samples=1000, transient=5000, thin=5,
                nChains=2, nParallel=2, verbose=500)
save(m2, file="m2.RData")
```

```
load("m2.RData")
mpost = convertToCodaObject(m2)
plot(mpost$Beta)
```









```
effectiveSize(mpost$Beta)
```

```
## B[(Intercept) (C1), sp1 (S1)]     B[forest. (C2), sp1 (S1)]
##                     1028.5910                      317.2271
```

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

```
##                                Point est. Upper C.I.
## B[(Intercept) (C1), sp1 (S1)]    1.000822   1.005198
## B[forest. (C2), sp1 (S1)]        1.000464   1.000483
```

```
summary(mpost$Beta)
```

```
##
## Iterations = 5005:10000
## Thinning interval = 5
## 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
## B[(Intercept) (C1), sp1 (S1)] 3.1819 0.2396 0.005357       0.007511
## B[forest. (C2), sp1 (S1)]     0.8011 0.2328 0.005205       0.013098
##
## 2. Quantiles for each variable:
##
##                                2.5%    25%    50%    75% 97.5%
## B[(Intercept) (C1), sp1 (S1)] 2.700 3.0318 3.1877 3.3423 3.641
## B[forest. (C2), sp1 (S1)]     0.328 0.6423 0.7998 0.9613 1.251
```

Compare the results to your previous results from the GLM exercise, and to a model fitted using maximum likelihood.

```r
library(glmmTMB)
m1 = glmmTMB(Eulaema_nigrita ~ forest. + (1|SA), REML=F, family="poisson", data=dat)
```

Write the results with emphasis on interpreting the parameter estimates and their credible intervals.