Statistics with Sparrows - many models, matrices, and some magic

Julia Schroeder

17/12/2017

Linear mixed models - Repeatability

Before we begin, we clear our workspace. I will never stop writing this. And I hope you will never stop doing this!

```
rm(list=ls())
setwd("~/Box Sync/Teaching/MagicalStats")
```

We will look at *repeatability*, in the unicorns. These unicorns have been captured repeatedly, and body mass has been measured repeatedly. What we do here is similar to what we've done yesterday, with BirdID as random factor. However, I will provide a more in-depth explanation of repeatabiilty, and with this, how to interpret random effects (of repeated measured on individuals). Another reason to do this hand out is to become more familiar with the package and function MCMCglmm. It's a very versatile package that allows flexible random effects structure. I love it.

Another way of saying that there is within-individual repeatability in a trait like wing length would be that individual bird have consistent wing length. They differ less between multiple measures than they differ from each other. Thus, the bird's wing length is *repeatable within individuals*. This is confusing, because we use the between-individual variance to calculate it. But it means that within individual birds, they are repeatable. We could also call it differences between individuals, and some authors do just that.

The statistical term is called *repeatablity*, or r, or "intraclass correlation coefficient", and it is important in several ways. It can be used to describe biological things, as we did here. It can however, also, be use to assess the quality of a method - to test for individual observer repeatability, as we did in the lecture.

The repeatability can be calculated in different ways. The simplest way is using the SS and MS of an ANOVA. A very good biologist and statistician, Kate "C" Lessells, published a paper on this, it is called "Unrepeatable repeatabilities: a common mistake" (Lessells and Boag 1987). This paper was rejected in many journals, and ultimately was published in a bird journal, "The Auk". It was published in 1987, and since has been cited 2381 times. It has been very successful in teaching generations of biologists how to do the correct statistics. I strongly urge you to read it, maybe best today, as now your knowledge on ANOVA is fresh, and you will understand it much easier!

However, for now, I will also explain how to calculate the repeatability. It is given as

$$r = \frac{s_A^2}{(s_W^2 + s_A^2)}$$

Where s_A is the among-group variance and s_W is the within-group variance. So what we really do here is we calculate the fraction of variance of the total variance ($s_W^2 + s_A^2$) that is explained by among-group differences. A over total. That's something really important for you to remember. We look at the % of variance explained by among-group differences. How can we do that? Remember how the ANOVA really only describes the variance partitions. We only used the sums of squares because the denominator canceled out. However, remember how calculating SS of among-groups was really complicated, because we had to weigh for the sample size of each group? Yes. That's what this paper is about - this weighing is not as easy as one migh think. It's easy for balanced group sizes. But if they are not balanced, you're in trouble. You can read this up in the Lessells and Boag paper I mentioned above, but for the record:

$$s_W^2 = MS_W$$

and

$$s_A^2 = \frac{MS_A - MS_W}{n_0}$$

The paper from Kate Lessells pointed out that many people used to just assume that we could just ignore n_0. That is wrong, because MS are not the variance unless we account properly for sample sizes of the groups. So we won't make this mistake. We know where the MS in the function come from, but what exactly is n_0 then? It is most often, actually not the sample size. It would be the sample size n if all group sizes were equal, say, we'd have 10 birds observed once, 10 twice, 10 thrice ect. That would be called a balanced dataset. However, much to our distress, in ecology and evolution, most datasets are far from balanced. Our datasets are often "dirty" and heterogeneous. As we saw earlier, we don't have balanced sample sizes for our BirdID ANOVA here. So we need to calculate n_0. It is calculated by a complicated function:

$$n_0 = [\frac{1}{a-1}[\sum_{i=1}^{a} n_i - (\frac{\sum_{i=1}^{a} n_i^2}{\sum_{i=1}^{a} n_i})]$$

This looks horribly complicated. And it is this only for the reason that there may be different sample sizes in groups. Ugh.

A much easier way of going about it is to use linear mixed models, where you can just pick your between-group variance estimate, and divide it by the total variance, which is the sum of the between-group variance plus the within (or among) group variance.

$$R = \frac{V_B}{V_B + V_A}$$

This is much easier, and coincidentially also better, for many reasons. We note that The result of R is a fraction. We can multiply it with 10 and then get a percentage. With that we know how much % of the total variance is explained by that random factor. Very cool. We can do that for all random factors, acutally, but it can only called repeatability if it's repeated measured. So, in that sense, we can have the repeatability within-individuals, or observer repeatability.

The concept of repeatability is an important one, and you should remember this for the rest of your life, as everything, really, in this course, especially when mixed models are involved. Over the time you spend here in Silwood, you will come across other methods of calculating the repeatability, but the principles stay the same: we look at the ratio of variance explained among vs between groups.

This is all very theoretical so far, but we'll get our hands dirty now. The idea here is to calculate the repeatability of something but use a different package for it. We need this to then run the bivariate models later today. If you find this difficult, do not despair! We will revisit that on day two of the Population genetics course in two weeks, but it's good if you know the basics until then!

First, we will start by estimating the repeatability of some morpholocial measurs in unicorns again. We will use the same dataset with repeated measures, however, this time, we will use a different method to estimate it. We will use Bayesian mixed models. And they are less scary than they sound!

First, we do our house keeping. Then we import the phenotypic data:

```
rm(list=ls())
setwd("~/Box Sync/Teaching/MagicalStats")

d<-read.table("DataForMMs.txt", header=T)
str(d)

## 'data.frame':    2000 obs. of  10 variables:
##  $ Individual    : int  2 2 7 2 2 2 2 2 4 4 ...
##  $ DateOfCapture : int  18 23 22 7 47 80 83 89 80 21 ...
##  $ LitterSize    : int  1 1 1 2 2 2 2 2 2 2 ...
##  $ Size          : num  -1.3 -3.052 -1.537 0.582 -1.404 ...
##  $ Hornlength    : num  1.421 1.53 2.075 1.21 0.322 ...
##  $ Bodymass      : num  -1.891 -2.955 0.348 -0.855 -1.781 ...
##  $ Glizz         : num  -1.3 -3.052 -1.537 0.582 -1.404 ...
##  $ SexualActivity: int  3 4 4 3 2 3 3 3 3 4 ...
##  $ Sex           : Factor w/ 3 levels "female","male",..: 1 1 1 1 1 1 1 1
1 1 ...
##  $ Family        : Factor w/ 10 levels "Buckwheat","Fluffytail",..: 4 4 4
4 4 4 4 4 4 4 ...
```

The first thing we do is findout if, and if so, how many, repeated measures we have for body mass.

```
table(table(d$Individual))
```

```
##
##  20
## 100
```

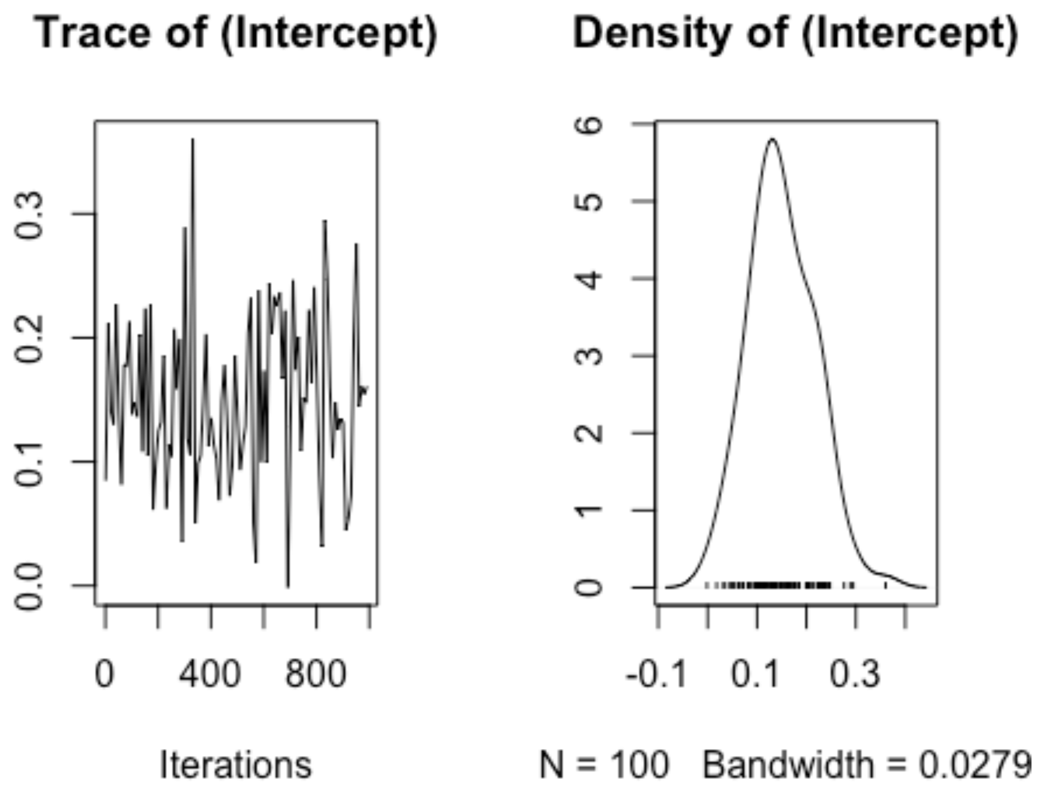
This gives us some idea how often unicorns have been measured (even though we might have known this from yesterday. The sample size is more than sufficient to partition variance in within- and between unicorn identity variance components.

We will use the package MCMCglmm (Hadfield 2010) to estimate the variance components (random effects). In this model, we are not interested in the fixed effects as we were when we ran simple linear models in the introductory statistics course. To tell R that we don't want to deal with fixed effects, we type ~1. That means only calculate the intercept. Then, we want R to estimate the random effects. In MCMCglmm you have to specify these by stating explicitely *random=*. This is nice, as it allows us to clearly see what is a random effect and what it fixed. So in the below model, we really only estimate variance due to BirdID – differences between individual birds, and variance within individual birds (residual variance, here called *units*, but not explicitely specified in the code). We then pass on the data argument, the number of iterations that we want the chain to run (*nitt*, more of this later), and the *burnin* (also more later).

```
require(MCMCglmm)

## Loading required package: MCMCglmm

## Loading required package: Matrix

## Loading required package: coda

## Loading required package: ape

m<-MCMCglmm(Bodymass~1, random=~Individual, data=d, nitt=1000, burnin=0)

##
##                         MCMC iteration = 0
##
##                         MCMC iteration = 1000
```

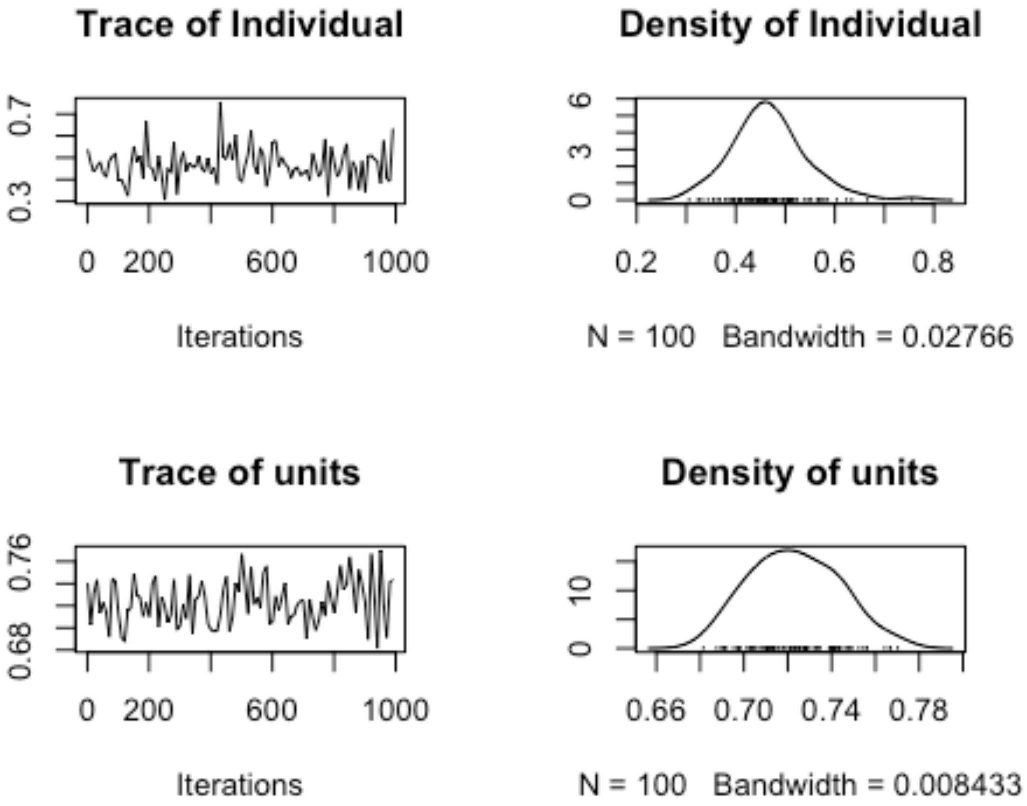
Now, before we continue, I will explain what is happening here. MCMCglmm estimates the solution of the mixed model using Bayesian methods. We have so far worked with ordinary least squares (in lectures), and (restricted) maximum likelihood (REML) methods (linear models). REML works well also for linear mixed models (GLMMs), and is fast and easy to use. However, for some applications, Bayesian Markov chain Monte Carlo (MCMC) Bayesian methods are better, as is most often the case when we are interested in more complex random effects models. If you remember maximum likelihood actually tries to maximum the likeihood that the parameter estimates fit the data best. Now, the equations that need solving for this are in some cases actually not easily analytically solvable. Worse, the confidence intervals are often not well defined, and neither are the degrees of freedom. The MCMC methods allow us to approximate the likelihood by using iterative Markov chains, where the model more or less uses a trial and error method for many different parameter estimate combinations. This is what the iterations mean: each iteration, the Markov chain draws randomly a sample fromt he joint posterior distribution. These distributions show potential parameter estimate values. We can look at those distributions by plotting the model. The posteriors for the variance components is stored in an element called *VCV*, and those for the fixed parameters in an element called *Sol* (equivalent to resid). These funny names shoulnd't confuse us any further, because it means the same:

```
plot(m$Sol)
```



Here we can see the trace of the posteriors through 1000 iterations, as we specified in the model (*nitt* = 1000). You can see that while the estimates change quite a bit, it is always somewhere between -0.1 and 0.4. As we did not specify any fixed effect, the intercept here gives us the mean body mass, which was 0.15. Normally we would run a MCMC model much longer than 1000 iterations, but more on that later. On the right hand side, these posteriors are plotted in a histogram-like density (histograms for continuous variables). The values they take on are not on the x-axis, and the mean of this histogram is in these models considered the parameter estimate. Using the density plot, it is clear that it is easy to work out the 95% confidence interval for the intercept. The little black dots on the x-axis indicate the frequency of how often certain values come up. Now we can look up the same plots for the random effects, or the variance components:

```
plot(m$VCV)
```

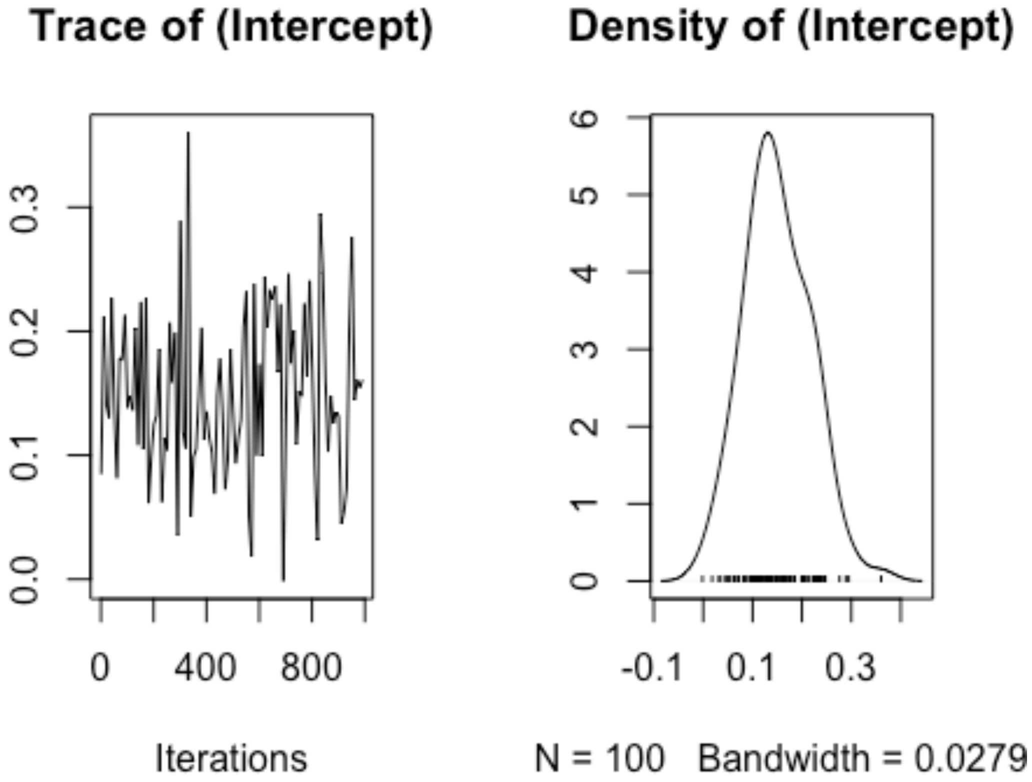


We get a plot for the Individual, and one for the residuals, which are here called *units*. As these are variance components, they cannot be smaller than zero. We can see that the Individual posteriors then zoomed in to remain around values of 0.4. The residuals are about 0.74. But as you can see from the plots on the left hand side, both started around 0.6-ish. This is becase the function assumes per default splits the variance between all components. In this case it didn't start exactly 50/50, but still then the model figured out that the fit is better when more variance is assigned to the residuals. This so-called *burn in* period is considered to be not so relevant for the solution of the model, so we usually discard it before examining the chain, by setting the burnin period to higher values. I set it to zero in this example to demonstrate this, but when you do analysis you NEVER should use a zero for the burnin. Also, the number of iterations (*nitt*) of 1000 is super low. Per default, MCMC runs for 13 000 iterations, and discards the first 3000 as a burn in. Let's try that:

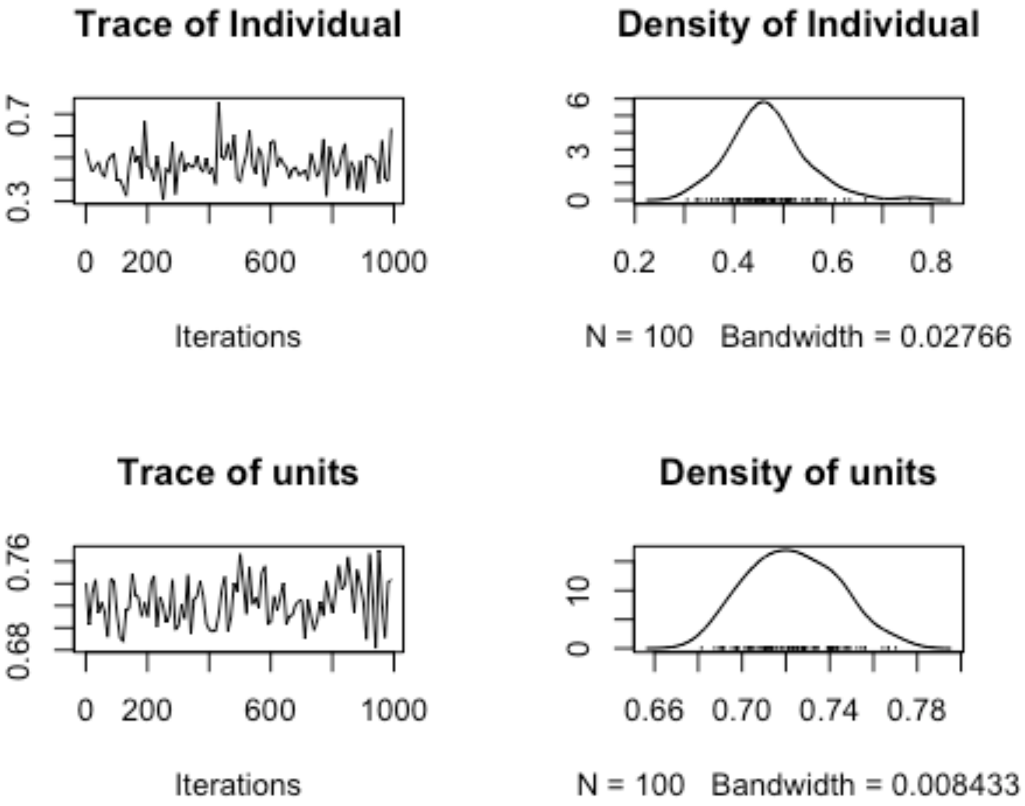
```
m1<-MCMCglmm(Bodymass~1, random=~Individual, data=d)

##
##                           MCMC iteration = 0
##
##                           MCMC iteration = 1000
##
##                           MCMC iteration = 2000
```

```
##
##                         MCMC iteration = 3000
##
##                         MCMC iteration = 4000
##
##                         MCMC iteration = 5000
##
##                         MCMC iteration = 6000
##
##                         MCMC iteration = 7000
##
##                         MCMC iteration = 8000
##
##                         MCMC iteration = 9000
##
##                         MCMC iteration = 10000
##
##                         MCMC iteration = 11000
##
##                         MCMC iteration = 12000
##
##                         MCMC iteration = 13000
```

```
plot(m$Sol)
```


Trace of (Intercept)
Density of (Intercept)
0.0
0.1
0.2
0.3
0
400
800
Iterations
0
1
2
3
4
5
6
-0.1
0.1
0.3
N = 100   Bandwidth = 0.0279

```
plot(m$VCV)
```



Now looking at the plots, we can see that the estimates have become much better, and the density plots much less skewed. We will run it a bit longer so the density plots get a bit more smooth, and use a longer burn in period:

Before we can inspect the model output, we have to run one more diagnostic. The Marcov chain is supposed to sample independently between two subsequent iterations. We need to make sure this is the case. To do that, we can compute the autcorrelation between successive samples for each variable:

```
autocorr(m$Sol)
```

```
## , , (Intercept)
##
##          (Intercept)
## Lag 0     1.00000000
## Lag 10   -0.05742172
## Lag 50    0.11848043
## Lag 100 -0.07101404
## Lag 500   0.09823962
```

With lag 1, the autocorrelation is 1, that is to be expected as we compare a posterior with itself. We can check lags between 10, 50, 100 and 500 iterations. The autocorrelation is ok if it is close to zero (0.01-0.01). We can live with these values though lag 10 is borderline. We now also check the ones for the random effects:

```
autocorr(m$VCV)

## , , Individual
## 
##           Individual       units
## Lag 0    1.00000000  0.03871832
## Lag 10  -0.10550984  0.11477754
## Lag 50   0.05118183  0.24350273
## Lag 100  0.05279626 -0.07944615
## Lag 500 -0.10571202  0.04224880
## 
## , , units
## 
##           Individual       units
## Lag 0    0.03871832  1.00000000
## Lag 10   0.07330386  0.03893685
## Lag 50  -0.20775740 -0.01388920
## Lag 100  0.06190797 -0.06234860
## Lag 500 -0.14459210  0.02308259
```

This is all very reasonable. If they'd be larger, you want to run the model longer.

Now we can look at the summary of the model:

```
summary(m1)

## 
##  Iterations = 3001:12991
##  Thinning interval  = 10
##  Sample size  = 1000 
## 
##  DIC: 5122.743 
## 
##  G-structure:  ~Individual
## 
##            post.mean l-95% CI u-95% CI eff.samp
## Individual    0.4621   0.3219   0.5967     1229
## 
##  R-structure:  ~units
## 
##       post.mean l-95% CI u-95% CI eff.samp
## units    0.7239   0.6762     0.77     1119
## 
##  Location effects: Bodymass ~ 1 
## 
##             post.mean l-95% CI u-95% CI eff.samp pMCMC  
## (Intercept)   0.15724  0.01929  0.30112     1273  0.04 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Let's inspect this output. The first line is clear, it tells us how many iterations, and which, have been used to generate this summary data. We ran the model for 13 000 iterations (it's discarded the last 9 because of the thin = 10, it will become clear in a second), and we discarded the 3000 first ones as burn in. The second line tells us that the thinning interval used was ten. This means that the function only stored every tenth posterior estimate. You can set this yourself to other values than the default 10 by specifying *thin=2* or *thin=100* or so in your model code. Think about to what you set this. Setting it to smaller values may increase the occurence of autocorrelation, but also gives you a larger sample size. Larger values mean your computer will have fewer data to store in its memory, which, for large models, can be important. So there are pros and cons. I usually go with the default value unless I have a reason not to. Thus, sampling every 10th value of 10 0000 iterations gives us a sample size of 1000 iterations that are summarized in this output. Then we get a DIC. That's the **D**eviance **I**nformation **C**riterion. We use it for Bayesian model selection, and it can help us decide which variables to keep in a model. Models with a smaller DIC are preferred, and we assume that differences between 3-10 DIC indicate a significant difference in model fit. Later more on that.

The next line informs us of the random effect structure. We have a simple random effect that is Indvidiual. Next come the parameter estimates of this random effect, or variance component. The posterior mean of the posterior density distribution is 0.47, with a 95% credibility interval (that's Bayesian lingo for confidence interval) of 0.34 - 0.61. The sample size from which this was estimated is also noted. This may change later when certain groups of random effects have different sample sizes.

Next, we get information on the error structure, or the residuals, or, units, and also 95CI.

Scrolling down, we get the fixed effects (termed location effects). We did not estimate any, to do that one uses a ~1 in the model code. Therefore, we only estimate an intercept. Again, we get the posterior mean, the 95CIs and an effective sample size. The 95CI is sufficient to consider this statistically significant as it does not span zero. However, a lot of people who were not used to work with confidence intervals complained, and thus the writer of this package gave in and provides now also a p-value and significance code, which are really unnesseccary, but each to their own.

Great. We understand this. Before we move on – do remember how these estimates are generated – with an iterative somewhat “random” walk, and that means that estimates may differ between runs. So if you find slight differences between runs/computers etc. that is not a cause for concern. However, the difference should be only slight quantitatively. If there are large differences something is wrong, and that should be investigated.

Now let's look back to the variance components, and our equation to calculate repeatability:

$$R = \frac{V_B}{V_B + V_A}$$

We extract the posterior means and 95% credibility intervals using this code:

```
posterior.mode(m1$VCV[,"Individual"])

##      var1
## 0.4055244

HPDinterval(m1$VCV[,"Individual"])

##          lower     upper
## var1 0.3218718 0.5967176
## attr(,"Probability")
## [1] 0.95

posterior.mode(m1$VCV[,"units"])

##      var1
## 0.7139695

HPDinterval(m1$VCV[,"units"])

##          lower     upper
## var1 0.6761719 0.7699572
## attr(,"Probability")
## [1] 0.95
```

To understand what this does let's have a quick look at one of these variables:

```
head(m1$VCV[,"Individual"])

## Markov Chain Monte Carlo (MCMC) output:
## Start = 3001
## End = 3061
## Thinning interval = 10
## [1] 0.4927711 0.5358282 0.4040207 0.4289735 0.4171732 0.4847011 0.4636655

str(m1$VCV[,"Individual"])

## Class 'mcmc'  atomic [1:1000] 0.493 0.536 0.404 0.429 0.417 ...
##   ..- attr(*, "mcpar")= num [1:3] 3001 12991 10

m1$VCV[,"Individual"]
```

So we are actually accessing each single parameter estimate of every 10th iteration (that's because the default setting is thin=10). So that's a long row of numbers! Very cool! And posterior.mode is just a function that calculates the mode of the posterior, and HPD interval gives us an interval of the posterior distribution (PD). The interval is, by default, set to 95%.

The neat thing about this is that now, we can calculate the total phenotypic variance of Bodymass of unicorns, and by using the actual posterior densities (the actualy vectors of values), we also get the 95% CI of the total phenotypic variation (and then, of repeatability!):

```
VT1=(m1$VCV[,"Individual"]+m1$VCV[,"units"])
posterior.mode(VT1)

##     var1
## 1.145624

HPDinterval(VT1)

##        lower    upper
## var1 1.03315 1.326642
## attr(,"Probability")
## [1] 0.95

var(d$Bodymass)

## [1] 1.17452
```

So, clearly out Bayesian estimate is pretty spot on to what we can caluclate with mean squares. However, the Bayesian estimate gives us a measure of precision - the 95CI, which we don't get with mean squares.

Clearly, the results don't match perfectly. Part of this is due to the inherent imprecisision of the Bayesian approach - if you run the model more often you will get different results, but they will always be within the 95CI, because that's what it means. In a sense, the Bayesian approach is more "realistic" as here a parameter estimate really is only that - an estimate - and the approach reflects that. We are forced to interpret a credibility interval rather than a single value, which better describes nature. Also, consider that the variance calculation does not consider the hierarchical nature of the data - that is another factor that greatly plays into this difference.

Now that we have all the ingredients, we can, finally, calculate the repeatability and even put a 95CI interval around it:

```
R1<-m1$VCV[,"Individual"]/(m1$VCV[,"Individual"]+m1$VCV[,"units"])
posterior.mode(R1)

##      var1
## 0.3913218

HPDinterval(R1)

##          lower     upper
## var1 0.3151702 0.4578234
## attr(,"Probability")
## [1] 0.95
```

The repeatability of bodymass in unicorns is about 38% , with a 95CI between 32% and 46%. Yay! Now if you feel somewhat comfortable with these ideas, you can move onto the bivariate models.

Also, because you made it this far, as a present from me I let you know that there’s a package that does all this for you and calculates repeatability, even for non-gaussian data, using link functions. A colleague of mine, who did his PhD on the Lundy sparrows, wrote it, and they published two highly cited papers on it (Nakagawa and Schielzeth 2010; Stoffel et al. 2017). That what can happen when you work on Lundy sparrows!

Hadfield JD. 2010. MCMC Methods for Multi-Response Generalized Linear Mixed Models: The MCMCglmm R Package. J. Stat. Softw. 33:1–22.

Lessells CM, Boag PT. 1987. Unrepeatable Repeatabilities: A Common Mistake. The Auk 104:116–121.

Nakagawa S, Schielzeth H. 2010. Repeatability for Gaussian and non-Gaussian data: a practical guide for biologists. Biol. Rev. 85:935–956.

Stoffel MA, Nakagawa S, Schielzeth H. 2017. rptR: repeatability estimation and variance decomposition by generalized linear mixed-effects models. Goslee S, editor. Meth. Ecol. Evol. 8:1639–1644.