glm to MCMCglmm

Kevin Healy

March 10th 2020

This is a short example of how to run linear models using a Bayesian approach. We will first start with a simple glm and then use MCMCglmm to run the same glm using a Bayesian approach. We will then run a more complex mixed model (glmm) using the standard frequentists approach before running one using a MCMCglmm. One of the strengths of MCMCglmm is its ability to run such mixed effects models with lots of different random terms, such as phylogeny, etc..

Installation

You should have already install the required packages from the prerequisites document that you can find on the workshops Github page (https://github.com/healyke/BES_Macro_workshop_Galway_2020). As an aside if you have not heard of GitHub before it is a great way to share code, build packages and use version control to back up your work. For more check out here

If you have successfully install all of the packages you can now load them.

```r
library(MCMCglmm)
library(lme4)
library(invgamma)
library(mulTree)
```

Data

For these exercises we will use some data that is part of the `MulTree` package. To get it just run

```r
data(lifespan)
```

This data file contains a subset of the data used in an analysis on the role of flying (volant) in the evolution of maximum lifespan in birds and mammals Link to paper. Note that these data have been log transformed, mean centred and expressed in units of standard deviation. The original lifespan data were taken from the Anage database. We will come back to why it is often useful to transform data into z-scores later but for now we will simply assume our data is well behaved. Lets have a look at it now.

```r
#data have been log transformed, mean centered and
#expressed in units of standard deviation.
head(lifespan_volant)
```

```
##                species    class  longevity       mass    volant
## 1 Dolichotis_patagonum Mammalia -0.1490041  1.0875446 nonvolant
## 2       Eidolon_helvum Mammalia  0.4686111 -0.2748337    volant
## 3      Elephas_maximus Mammalia  2.1071286  3.1220340 nonvolant
## 4         Equus_asinus Mammalia  1.6128024  2.0352764 nonvolant
## 5     Equus_burchellii Mammalia  1.2962194  2.2295299 nonvolant
## 6       Equus_caballus Mammalia  1.9001076  2.2548716 nonvolant
```

GLMs

Let's first start off running a simple glm for a subset of data for mammals

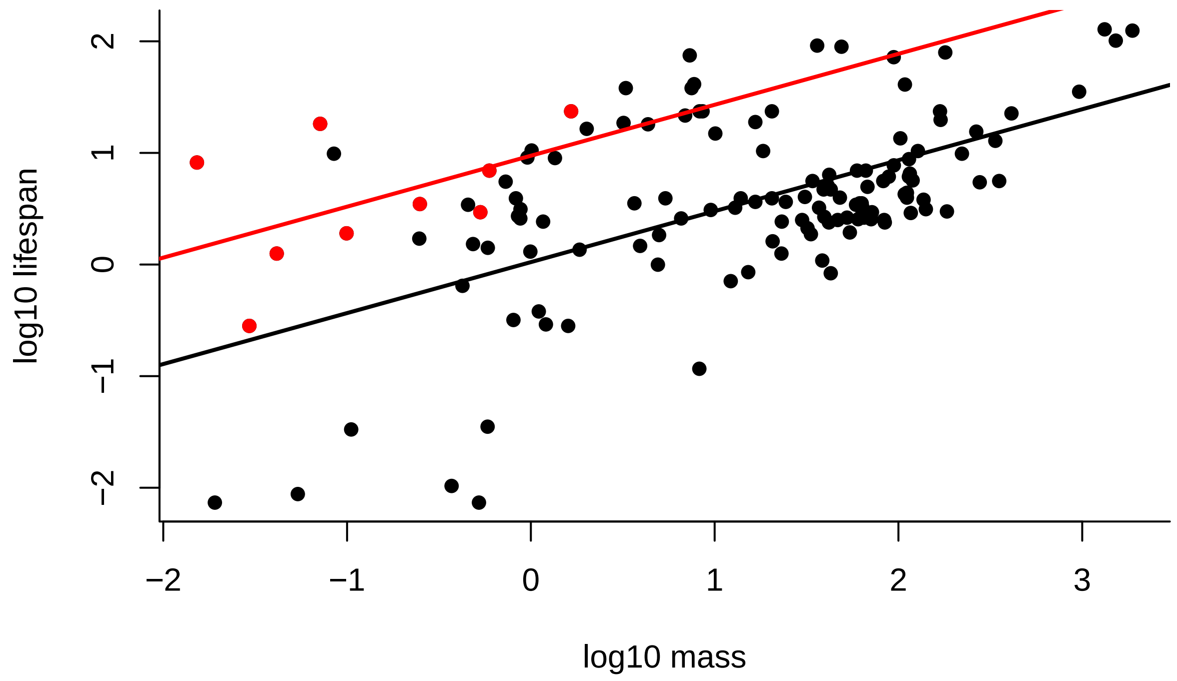
```
#subset for mammals
lifespan_mammals <- lifespan_volant[lifespan_volant$class == "Mammalia",]

#### and run a simple glm
glm_mod <- glm(formula = longevity ~ mass + volant,
               family = "gaussian",
               data = lifespan_mammals)


summary(glm_mod)
```

```
##
## Call:
## glm(formula = longevity ~ mass + volant, family = "gaussian",
##     data = lifespan_mammals)
##
## Deviance Residuals:
##      Min        1Q    Median        3Q       Max
## -2.02569  -0.38641  -0.07613   0.41818   1.46075
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept)   0.02144    0.09241   0.232 0.816885
## mass          0.45655    0.05844   7.812  1.6e-12 ***
## volantvolant  0.95325    0.25568   3.728 0.000286 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for gaussian family taken to be 0.423457)
##
##     Null deviance: 81.317  on 133  degrees of freedom
## Residual deviance: 55.473  on 131  degrees of freedom
## AIC: 270.09
##
## Number of Fisher Scoring iterations: 2
```

We can plot the results

```
#simple plots
plot(longevity ~ mass, data = lifespan_mammals, pch = 16, bty = "l",
                       xlab = "log10 mass", ylab = "log10 lifespan")

#add in the volant species as red
points(lifespan_mammals[lifespan_mammals$volant == "volant","longevity"] ~
        lifespan_mammals[lifespan_mammals$volant == "volant","mass"],
        col = "red", pch = 16)
```

```
#add in the nonvolant regression line
abline(glm_mod$coefficients[1],
       glm_mod$coefficients[2],
       lwd = 2)

#add in the volant regression line
abline(glm_mod$coefficients[1] + glm_mod$coefficients[3],
       glm_mod$coefficients[2], lwd = 2, col = "red")
```



Most people will be familiar with lm and glm models so we won't spend any more time here. Now lets run a standard linear model in MCMCglmm

MCMCglmm

So far we have fitted a very simple glm, now we will fit a linear model using the MCMCglmm package.

Since we are using a Bayesian approach we will need to set up the priors. In most cases we want to use a non-informative prior that doesn't influence the estimated posterior distribution. We are basically saying that we don't know anything about the expected values for our parameters. That is, we have no prior information.

To give priors for MCMCglmm we need to make an object that is in a list format that includes terms of B (fixed effect), R (residual terms) and G (random terms which we will come to later).

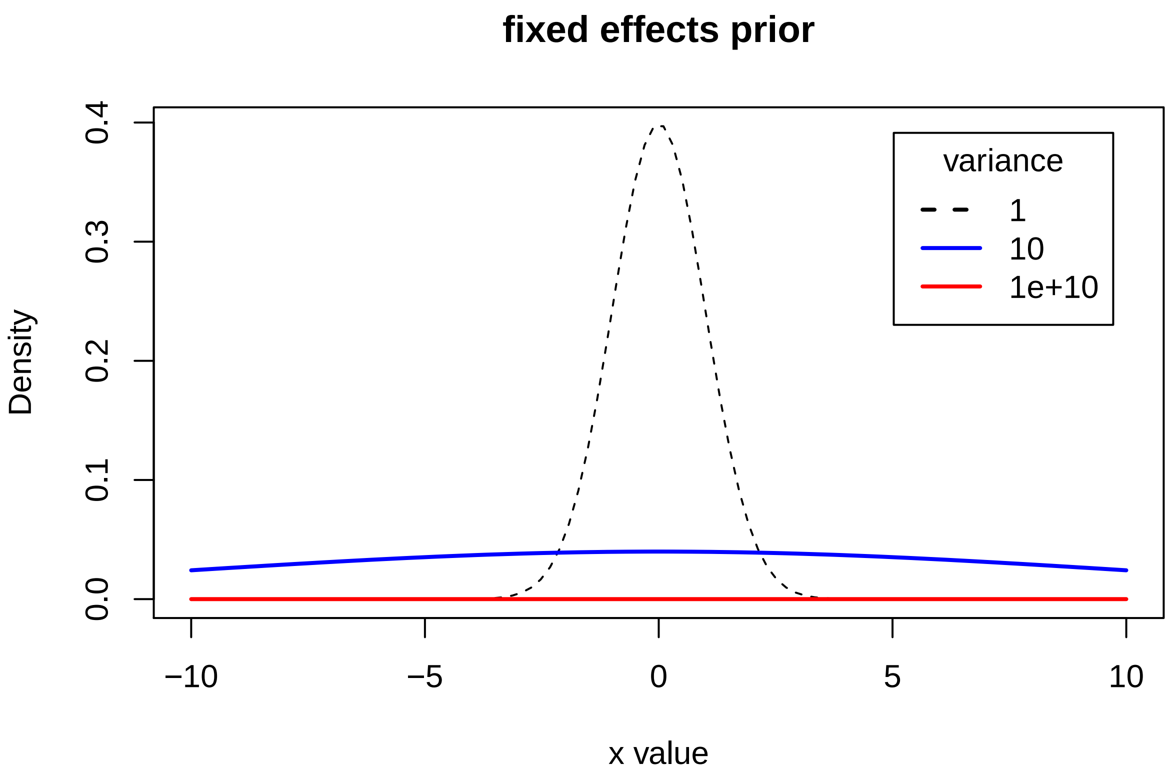
For now let's build a prior with just a fixed term and a residual term.

```
prior <- list(B = list(mu= diag(3)*0, V=diag(3)*1e+10),
              R = list(nu=0.002, V=1))
```

For fixed effects (B) the terms V and mu give the variance and mean of a normal distribution. Here we set mu as 0 and the variance as a large number to make these priors uninformative. Since we have three fixed terms (two intercepts and one slope) we can use the diag function to create a matrix to store a prior for each. MCMCglmm will automatically set non-informative priors for fixed terms and so you generally only see R and G set in on-line examples.

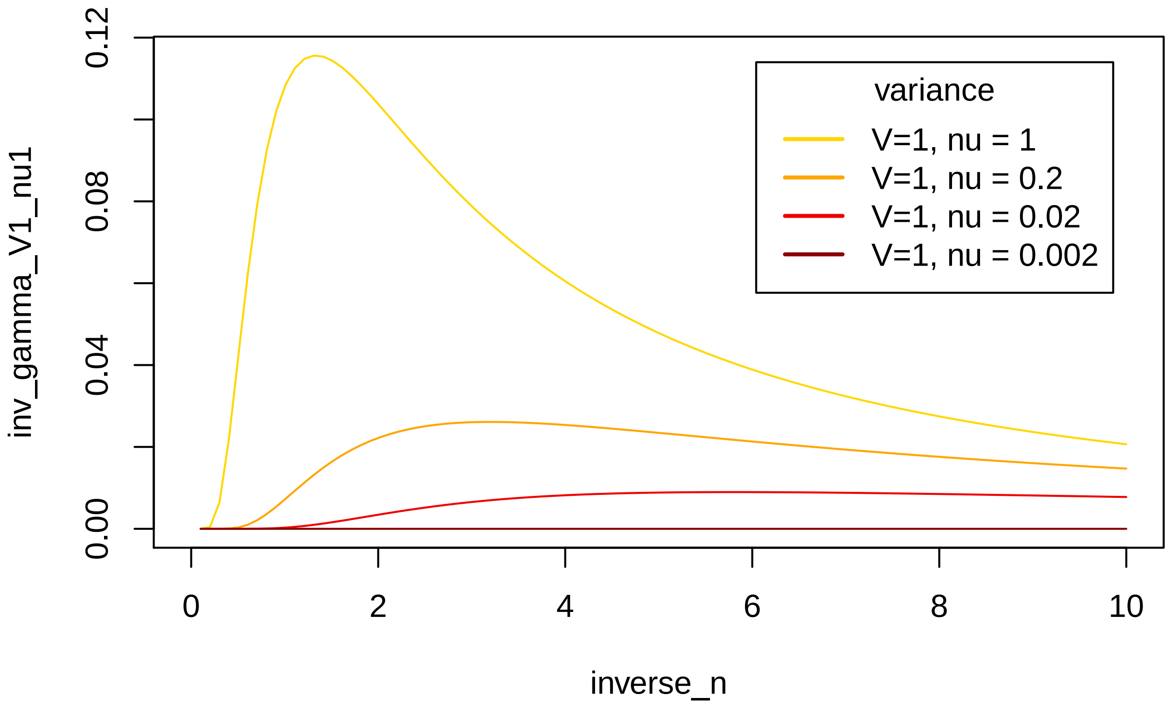
If we plot it out, we can see that higher values for V give less informative priors.

```r
#create some normal distributions over some range x
x <- seq(-10, 10, length=100)

#when V = 1
hx_1 <- dnorm(x, 0,1 )

#when V = 10
hx_10 <- dnorm(x, 0,10 )

#when V = 1e+10
hx_1e10 <- dnorm(x, 0,1e+10 )

plot(x, hx_1,
     type="l",
     lty=2,
     xlab="x value",
     ylab="Density",
     main="fixed effects prior")

lines(x, hx_10, lwd=2, col="blue")

lines(x, hx_1e10, lwd=2, col="red")

labels <- c("1","10","1e+10")
legend("topright", inset=.05, title="variance",labels,
       lwd=2, lty=c(2, 1, 1), col=c("black", "blue", "red"))
```



For the variance terms, such as the residual term, we need to make sure that the distribution is bounded at zero as the variance term needs to be positive. To do this MCMCglmm uses an inverse-Gamma distribution. In MCMCglmm this is described by two parameters `nu` and `V`. These terms are related to the shape (alpha) and scale (beta) parameters on an inverse-Gamma with `alpha = nu/2`, and `Beta = (nu*V)/2`. As we don't want our estimates to be heavily influenced by our prior we will use weakly informative prior values such as described as `V = 1` and `nu = 0.002`. (For more on priors the see course notes)

```
#create an inverse distributions over some range x
inverse_n <- seq(0, 10, length= 100)

#when V = 1, nu = 1
nu_1 <- 1
V_1 <- 1
inv_gamma_V1_nu1 <- dinvgamma(inverse_n,
                              shape = nu_1/2,
                              scale = nu_1*V_1/2)

#when V = 1
nu_05 <- 0.5
V_1 <- 1
inv_gamma_V1_nu05 <- dinvgamma(inverse_n,
                              shape = nu_05/2,
                              scale = (nu_05*V_1)/2)

#when V = 1
nu_02 <- 0.3
V_1 <- 1
inv_gamma_V1_nu02 <- dinvgamma(inverse_n,
                              shape = nu_02/2,
                              scale = (nu_02*V_1)/2)

#when V = 1
nu_0002 <- 0.002
V_1 <- 1
inv_gamma_V1_nu0002 <- dinvgamma(inverse_n,
                              shape = nu_0002/2,
                              scale = (nu_0002*V_1)/2)



plot(inverse_n,
     inv_gamma_V1_nu1, type="l",
     col = "gold")

lines(inverse_n, inv_gamma_V1_nu05,
      col = "orange")
lines(inverse_n, inv_gamma_V1_nu02,
      col = "red2")
lines(inverse_n, inv_gamma_V1_nu0002,
      col = "darkred")


labels_nu <- c("V=1, nu = 1",
            "V=1, nu = 0.2",
            "V=1, nu = 0.02",
            "V=1, nu = 0.002")

legend("topright",
       inset=.05,
       title="variance",
       labels_nu,
       lwd=2, lty=c(1, 1, 1, 1),
       col=c("gold", "orange", "red2", "darkred"))
```



Next we need to decide on the parameters relating to running the mcmc chain in the model. We need to include how many iterations we want to run the chain for (nitt), the burnin we want to discard at the start of the chain (burnin) and also how often we want to sample and store from the chain (`thin`). We discard a burnin as we don't want the starting point of the chain to over-influence our final estimates. For now let's just use a burnin of 1/6 of the `nitt`, just to be safe. The thinning is used to help reduce autocorrelation in our sample, how much you use often depends on how much autocorrelation you find, more on this later.

To save time we will only run this model over 12000 iterations (However, much larger `nitt` is often required).

```
#no. of interations
nitt <- c(12000)
#length of burnin
burnin <- c(2000)
#amount of thinning
thin <- c(5)
```

Now we can run the model using our data. Let's run a model similar to our first glm

```
mod_mcmc_glm <- MCMCglmm(fixed =  longevity ~ mass + volant,
                     family="gaussian",
                     data = lifespan_mammals,
                     nitt = nitt,
                     burnin = burnin,
                     thin = thin,
                     prior = prior)
```

```
##
##                        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
```

First congratulations, you have just run a Bayesian glm model. I really can be that simply however it is important to try and understand what is going on so we will spend some time going through it.

The first thing you will notice is that as the model runs we see the iterations print out. These chains can take some time to run, depending on the model, however, since we only ran our chains for 12000 iterations it doesn't take long here.

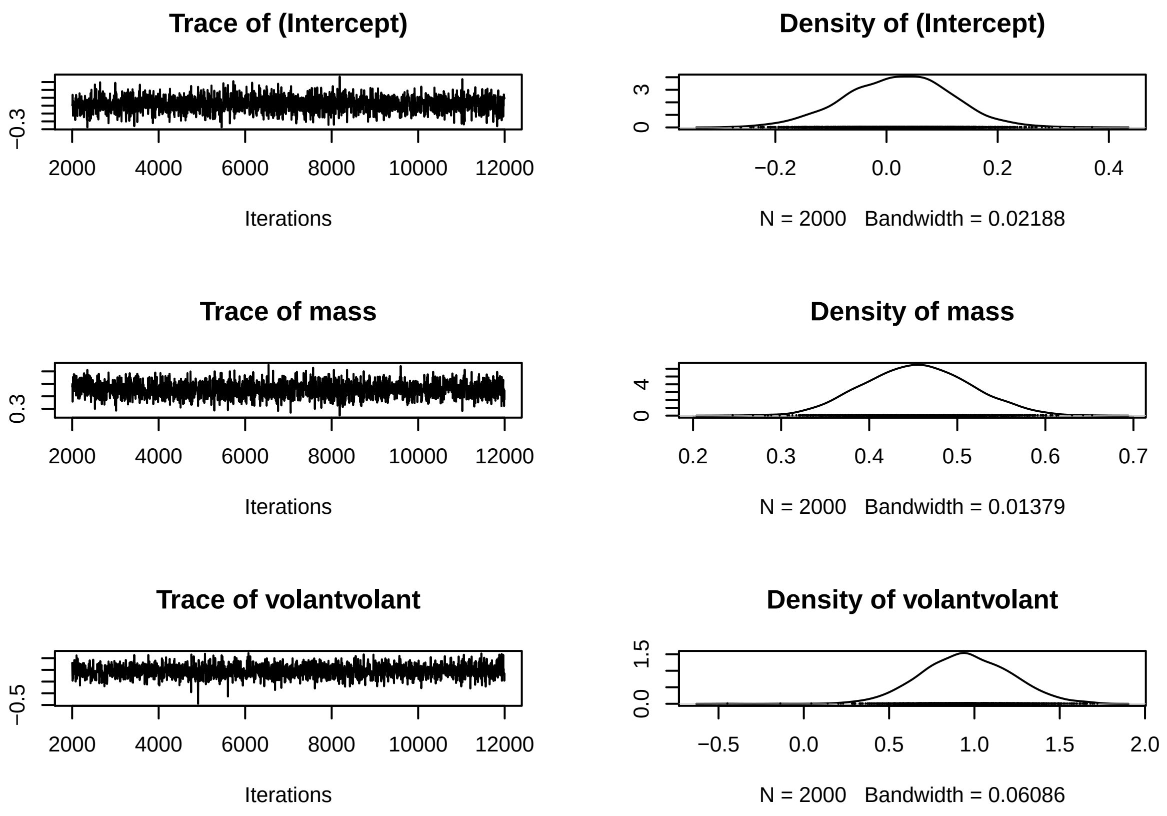
We will come back to checking how well the model ran later but for now we will go ahead and assume everything is OK.

```
summary(mod_mcmc_glm)
```

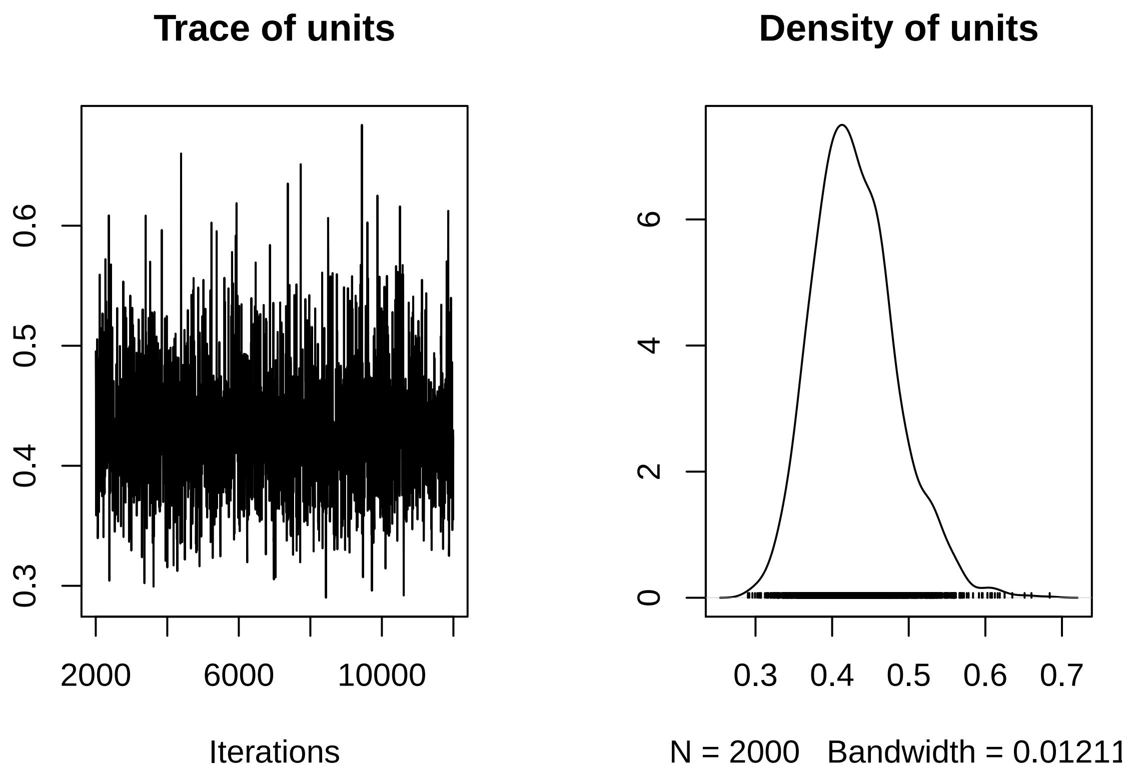
```
##
##  Iterations = 2001:11996
##  Thinning interval  = 5
##  Sample size  = 2000
##
##  DIC: 270.1433
##
##  R-structure:  ~units
##
##       post.mean l-95% CI u-95% CI eff.samp
## units    0.4301   0.3357   0.5424     2000
##
##  Location effects: longevity ~ mass + volant
##
##              post.mean l-95% CI u-95% CI eff.samp  pMCMC
## (Intercept)    0.02097 -0.15936  0.21162     2000  0.801
## mass           0.45483  0.34210  0.57106     2000 <5e-04 ***
## volantvolant   0.95159  0.45624  1.47523     2000  0.002 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

First off we can find the estimates for the fixed factors are under the Location effects section (Notice the similarity to our glm model). These estimates give the mean and the lower and higher 95% credible interval (CI) of the posterior distribution for each of the parameters we estimated. These posterior distributions are stored in the model as `model$Sol` for the fixed effects and `model$VCV` for the variance terms, which is just the residual term in this case. Lets plot them out and have a look.

```
#plot the fist fixed term, the intercpet.
plot(mod_mcmc_glm$Sol)
```



```
#plot the fist variance term, the residual error term.
plot(mod_mcmc_glm$VCV)
```



On the right hand side of the plots is the posterior distributions for each of the terms. On the left side of these plots are the traces of the mcmc chain for each estimate (More on these when we check out models). Notice that the posterior distributions are just distributions and so you can treat them like that.

For example, we can simply calculate the median of the intercept as below.

```
median(mod_mcmc_glm$Sol[,1])
```

```
## [1] 0.02311392
```

We can also calculate the mode and other credibility intervals for each of the posterior distributions. For example, we can calculate the mode of the slope as:

```
hdr(mod_mcmc_glm$Sol[,2])$mode
```

```
## [1] 0.4544192
```

and the 50%, 95% and 99% credibility intervals

```
hdr(mod_mcmc_glm$Sol[,2])$hdr
```

```
##          [,1]      [,2]
## 99% 0.3090647 0.6037325
## 95% 0.3401928 0.5697843
## 50% 0.4130217 0.4946958
```

So how do we know if our estimates are "significant"? From a philosophical point of view there is no significant when we talk about Bayesian statistics. Its simple a case of looking at the posterior distributions and deciding based on that.

However, from a pragmatic point of view people still like to see significance and p-value equivalents. The `pMCMC` value can be treated like a p-value, although its technically not. The approach I use, and the reason I z-scored my data (mean centring and dividing each continuous variable by it standard deviation), is that I can now just look at my posterior distributions and ask if both 95% credibility intervals are above or below zero.

We also have the DIC which is a Bayesian version of AIC. Like AIC it is a measure of the trade-off between the "fit" of the model and the number of parameters, with a lower number indicating a better fit.

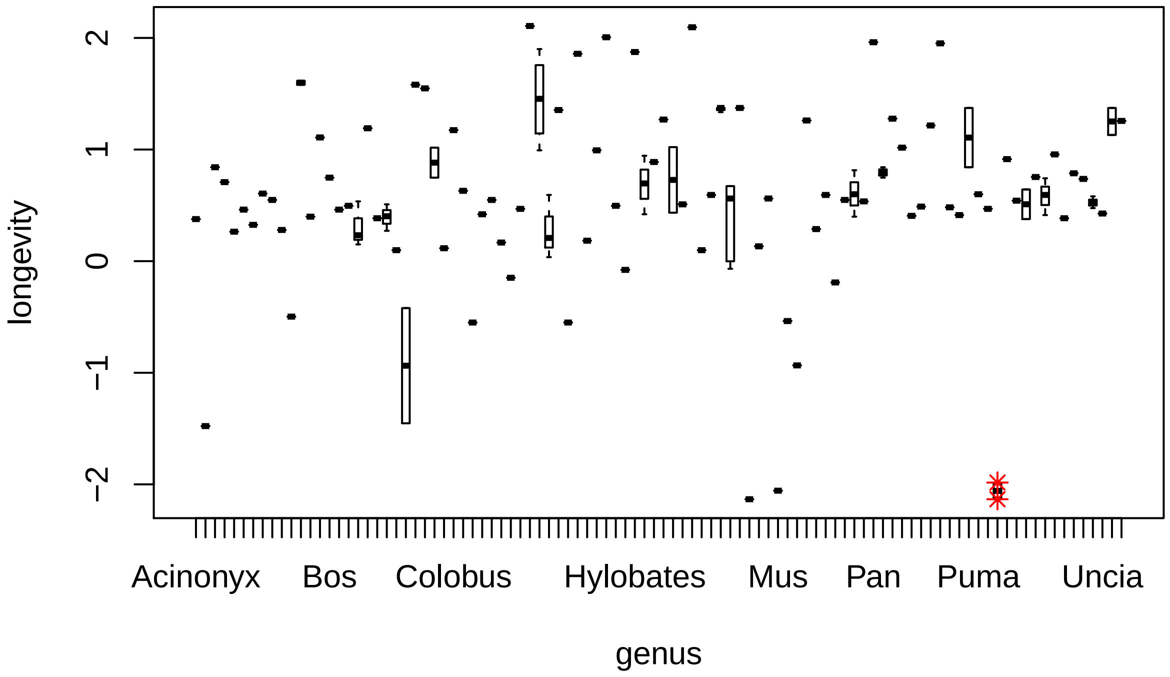
Putting the m in MCMCglmm

So far we have just run a simple linear models. The power of MCMCglmm is its flexibility in running models with random terms in them. So lets have a look at some mixed models with random terms. first

One reason we might want to use a mixed effects model is that we might want to try to control for the structure of our data. For example, maybe all the data points are not fully independent with some data points more likely to have values closer to other ones. We can imagine such a case in our longevity data above were two species from the same genus might show more similar lifespans in comparison to other species.

Let's plot it out and have a look

```
#use the gsub function to create a vector of the genus for each species
genus <- (gsub("_.*","", lifespan_mammals$species))

#bind it back into the database
lifespan_mammals <- data.frame(lifespan_mammals, genus = genus)

#plot out lifespan ~ genus to get an idea of whether genus is structured
#randomly
plot(longevity ~  genus, data = lifespan_mammals)

#lets highlight one of the outlier groups, the rats.
points(lifespan_mammals[lifespan_mammals$genus ==  "Rattus","longevity"] ~ c(85,85),
       col = "red",
       pch = 8)
```



This looks to me like genus is something I would like to control. For example, the genus Rattus can be seen to generally have a low longevity. However, I might not really be interested in which groups are different and for practical reasons I don't want to fit every single group as a fixed factor. Hence I could include it as a random term using a lmer model which is used to fit mixed models using a maximum likelihood approach.

```
#Lets fit our model. For lmer the random term is fitted using (1|genus)
#to indicate that you want to fit seperate intercepts
#to each of group in your random term.
lmer_mod  <-  lmer(longevity ~ mass + volant  + (1|genus), data = lifespan_mammals)
summary(lmer_mod)
```

```
## Linear mixed model fit by REML ['lmerMod']
## Formula: longevity ~ mass + volant + (1 | genus)
##    Data: lifespan_mammals
##
## REML criterion at convergence: 220.9
##
## Scaled residuals:
##     Min      1Q  Median      3Q     Max
## -2.0997 -0.2667 -0.0240  0.3312  1.8262
##
## Random effects:
##  Groups   Name        Variance Std.Dev.
##  genus    (Intercept) 0.37772  0.6146
##  Residual             0.05953  0.2440
## Number of obs: 134, groups:  genus, 98
##
## Fixed effects:
##               Estimate Std. Error t value
## (Intercept)  -0.01585    0.10366  -0.153
## mass          0.48637    0.06365   7.641
## volantvolant  1.00386    0.27950   3.592
##
## Correlation of Fixed Effects:
##             (Intr) mass
## mass        -0.746
## volantvolnt -0.534  0.496
```

Like before we see estimates for each of the fixed effects which look similar to our glm model. The next section that we are interested in is the random effects were we see the terms genus and Residual. Our residual term is acting as in the glm model telling how much of the variance is unexplained while the genus term tells us how much of the variance is due to variance associated with the genus groupings. We can see here for example that genus accounts for more variation than our random term after accounting for our fixed effects.

I will leave lmer models here as this was just to demonstrate using a methods that many of you might be more familiar with. For now lets move on to the main event with MCMCglmm.

Since we can run a simple linear MCMCglmm lets add a random term of genus in the example above. Like before we need to set up the prior however we will let the model estimate the fixed effects this time. To add a random term we now add a `G structure` that acts just like the other random variance term and is defined using `nu` and `V`.

```
prior_mixed <- list(G = list(G1 = list(nu=0.002, V=1)),
                    R = list(nu=0.002, V=1))

prior_mixed
```

```
## $G
## $G$G1
## $G$G1$nu
## [1] 0.002
##
## $G$G1$V
## [1] 1
##
##
##
## $R
## $R$nu
## [1] 0.002
##
## $R$V
## [1] 1
```

Like before we need to set the burn-in, iteration length and thinning. We will keep it very short so that the model will run fast and we can check this later.

```
#number of iterations
nitt_m <- 12000

#burnin length
burnin_m <- 1

#thinning
thin_m <- 1
```

Like before we set our model up with the fixed effects (i.e the main model), the family for the error term (normal so we use "gaussian"), the number of iterations (nitt_m), burn-in, thinning and prior.

We now also include the random term in the model in the section `random= ~`. I our case its the genus variable we created. Note also that we have included `rcov= ~units`. This is the residual term and was automatically included in the previous MCMCglmm model we already ran. Now we explicitly define it as we are including other variance terms.

With these extra random terms we can now run our mixed effects model.

```
mod_mcmc_mixed <- MCMCglmm(fixed = longevity ~ mass + volant,
                           rcov=~ units,
                           random= ~ genus,
                           family="gaussian",
                           data = lifespan_mammals,
                           nitt = nitt_m,
                           burnin = burnin_m,
                           thin = thin_m,
                           prior = prior_mixed)
```

```
##
##                        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
```

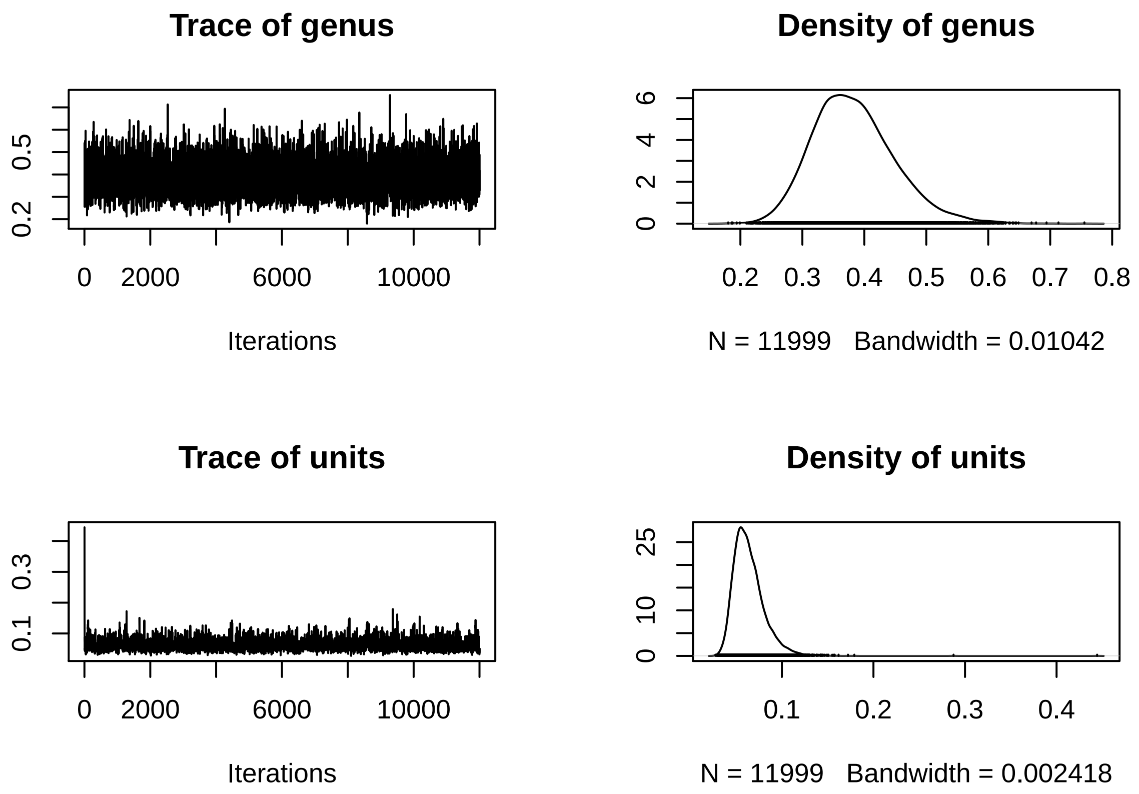
```r
summary(mod_mcmc_mixed)
```

```
##
##  Iterations = 2:12000
##  Thinning interval  = 1
##  Sample size  = 11999
##
##  DIC: 91.68768
##
##  G-structure:  ~genus
##
##       post.mean l-95% CI u-95% CI eff.samp
## genus    0.3828   0.2616   0.5145     5861
##
##  R-structure:  ~units
##
##       post.mean l-95% CI u-95% CI eff.samp
## units    0.0639   0.0371  0.09782     1710
##
##  Location effects: longevity ~ mass + volant
##
##              post.mean l-95% CI u-95% CI eff.samp    pMCMC
## (Intercept)   -0.01426 -0.21630  0.18727    11612 0.893574
## mass           0.48595  0.36310  0.61136    11560  < 8e-05 ***
## volantvolant   1.00070  0.46385  1.55558    11999 0.000333 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Our model looks just like our previous model apart from those extra random terms. Before we discuss those terms compare the fixed terms of this model to the standard glm as an exercise. Plot out the posterior distributions and calculate the median and mode for each of the fixed factors.

Now lets check out the random terms. As we saw before the residual term can be found using `model$VCV`. This is also were we will find all the variance terms (VCV stands for variance co-variance)

```r
#plot the fist variance term, the residual error term.
plot(mod_mcmc_mixed$VCV)
```

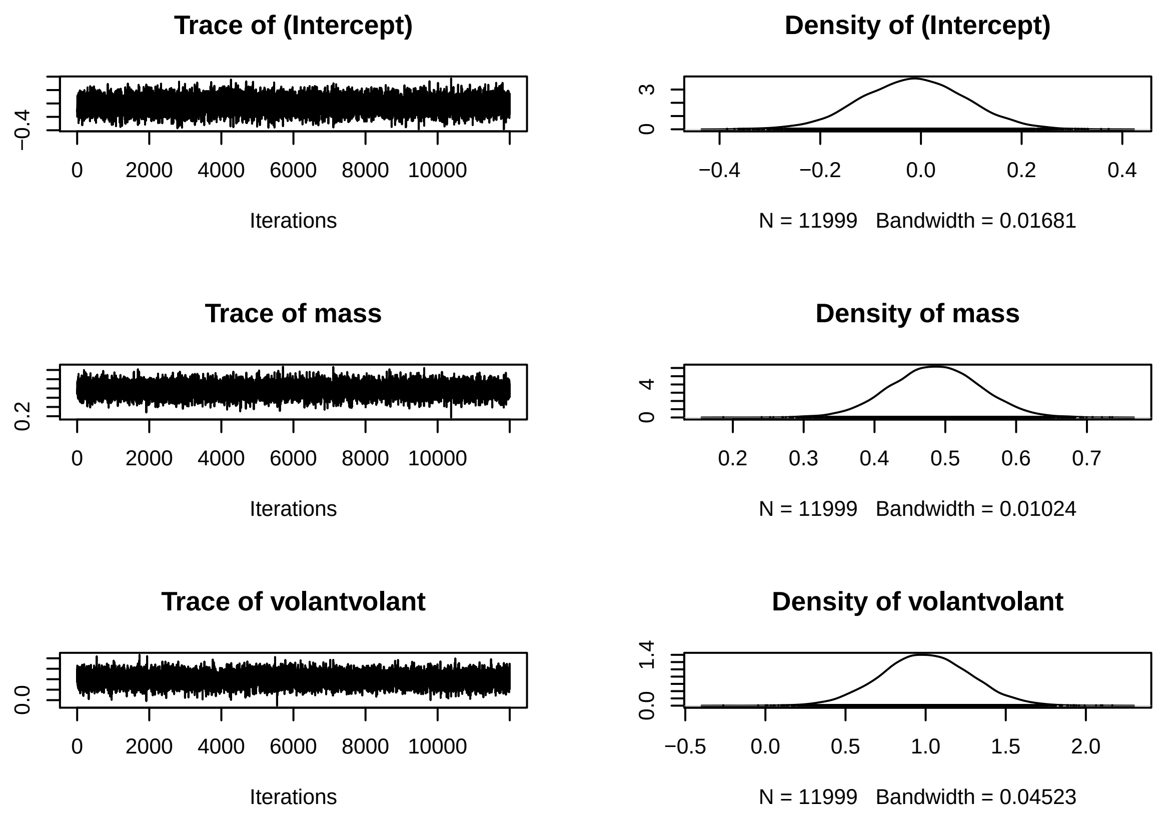


As we can see there genus does indeed account for some variation. While we will look at some more complicated models we have basically covered how to run a simple mixed effect model. However, when running these models its important to check that they ran correctly.

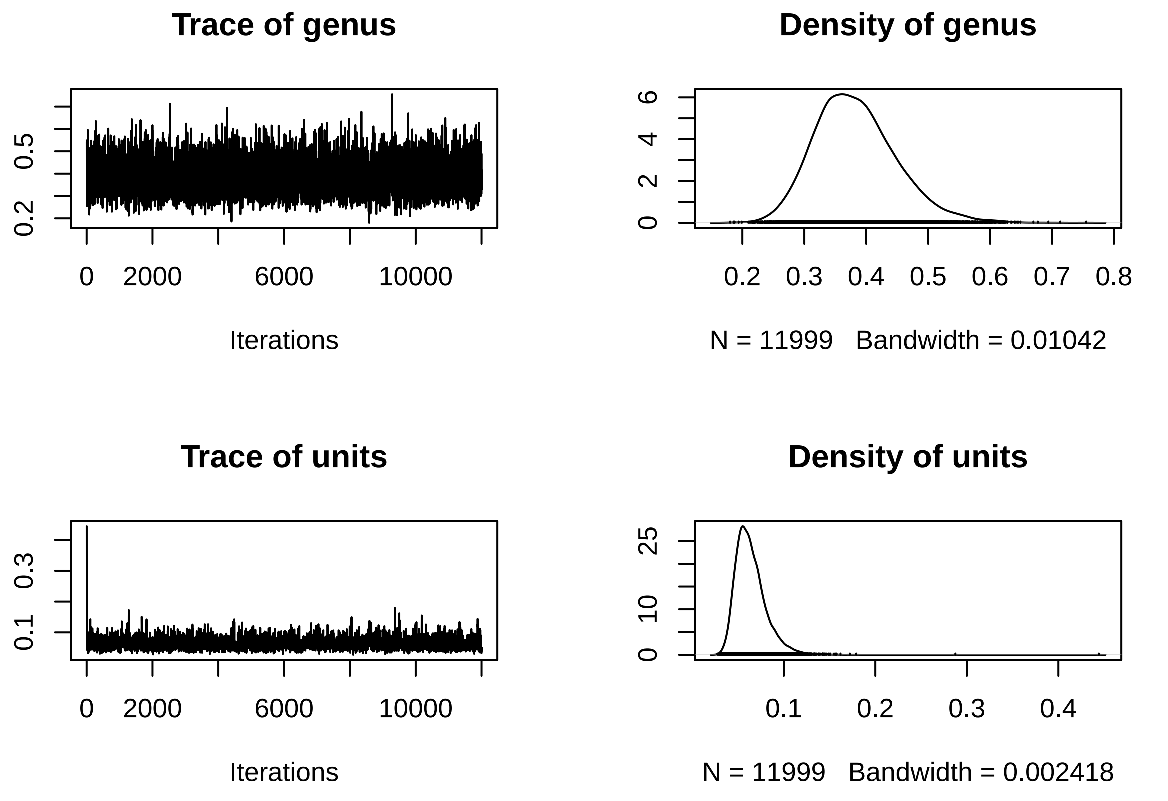
Checking you MCMCglmm Model

When running an MCMCglmm one of the first things you need to check is the how your MCMC chain ran. Remember that an MCMC chain is a process exploring value space so we need to see that it is doing just that and also that it found some sort of peak value. The first thing we do is to look at the trace. Lets check our mixed effects model.

```r
#plot the fist fixed term, the intercpet.
plot(mod_mcmc_mixed$Sol)
```

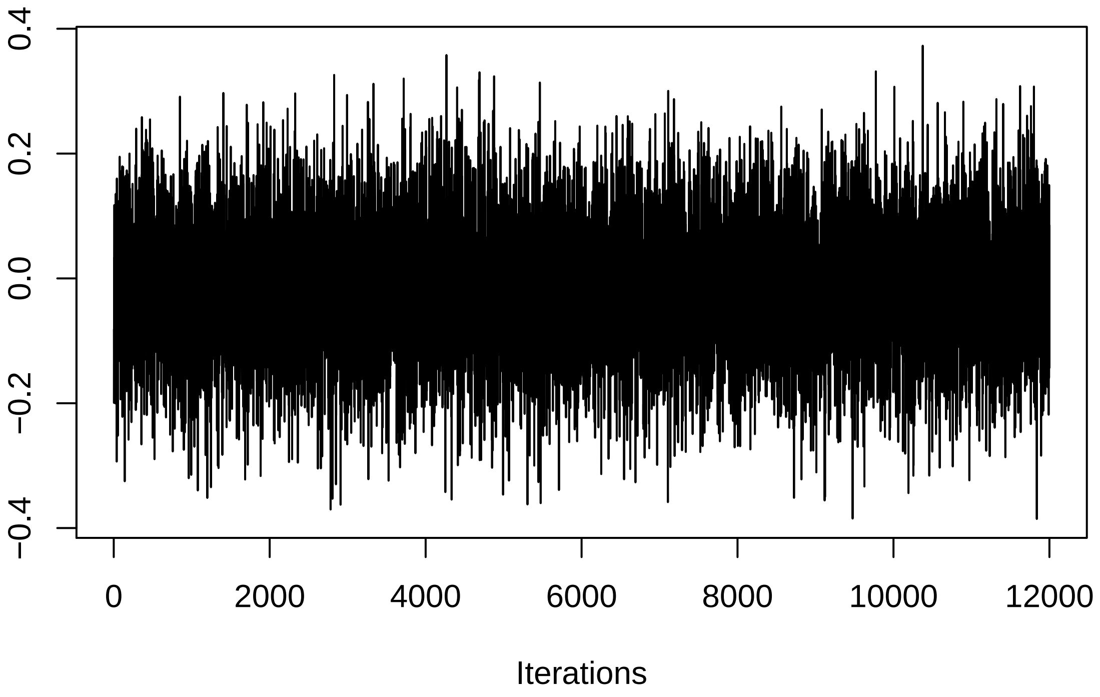


```r
#plot the fist variance term, the residual error term.
plot(mod_mcmc_mixed$VCV)
```



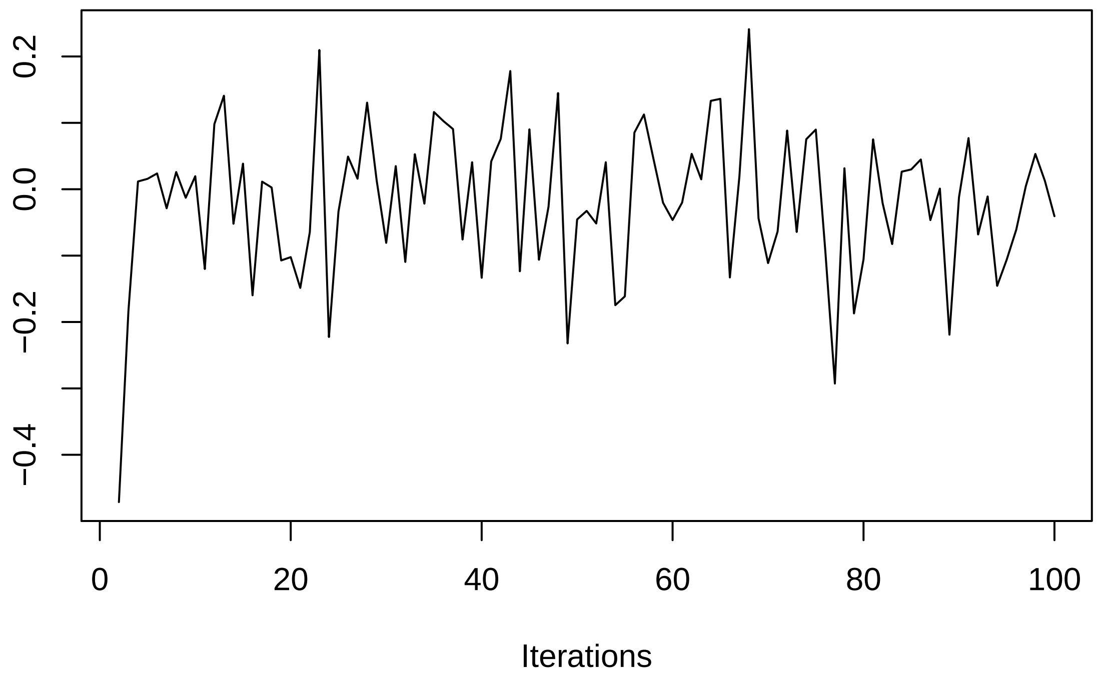
On the left side of these plots are the traces of the mcmc chain for each estimate. We can focus in on any one trace using `traceplot()`.

```r
#trace of the intercept MCMC chain

traceplot(mod_mcmc_mixed$Sol[,1])
```

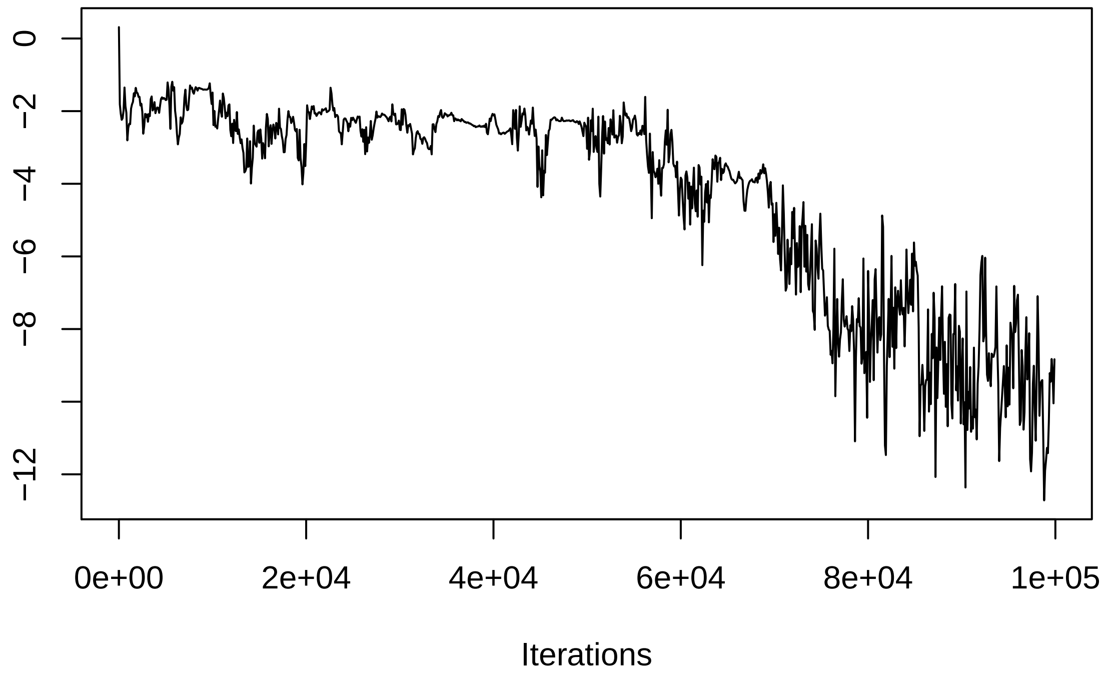


What we want to see in these trace plots is "hairy caterpillars". That is a trace with no obvious trend that is bouncing around some stable point. This means that the trace is exploring value space but also centring around one value. The posterior distribution is simply just a case of adding up this trace so the distribution can have any shape. What we want to see is that is is starting to agree on one value.

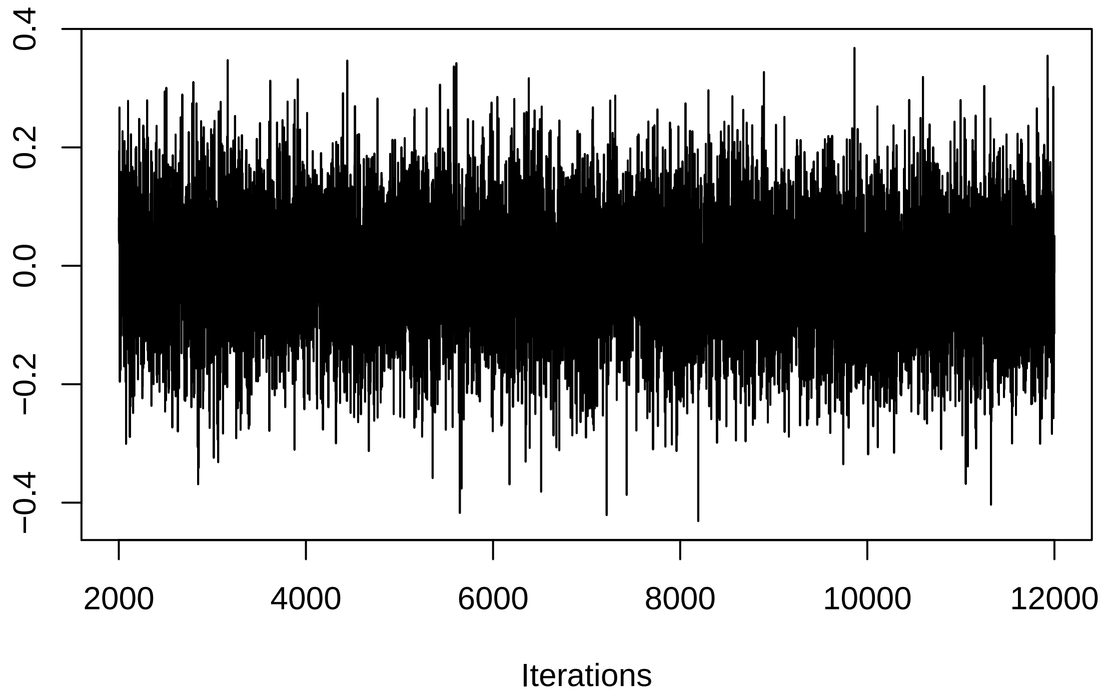
What we don't want to see in the trace plots can be demonstrated if we only run a model over a very short chain (itt == 1000) or more difficult model fit (we will see these more difficult models later).



Just as an example the below model is even worse.



Notice in the example above that without a burnin the start of trace is well outside the area that the chain is converging towards. This is why we use a burn-in. There is no hard rule but generally I use 1/6 of the number of total iterations as a burn-in. The other easy remedy is to run it for a longer chain.



Looking at these traces what we typically want is for our fixed effects to have general normal posterior distributions. The random effects will often have more log-normal distributions as they are bounded at zero but check that the chain is still exploring and not stuck at zero. This effect of getting stuck is referred to as poor mixing. More on this later.

So far in our simple model everything looks good visually, however we also want to check the level of auto-correlation in these traces. We can do this using autocorr.diag() which gives the level of correlation along the chain between some lag sizes.

```
autocorr.diag(mod_mcmc_good_fit$Sol)
```

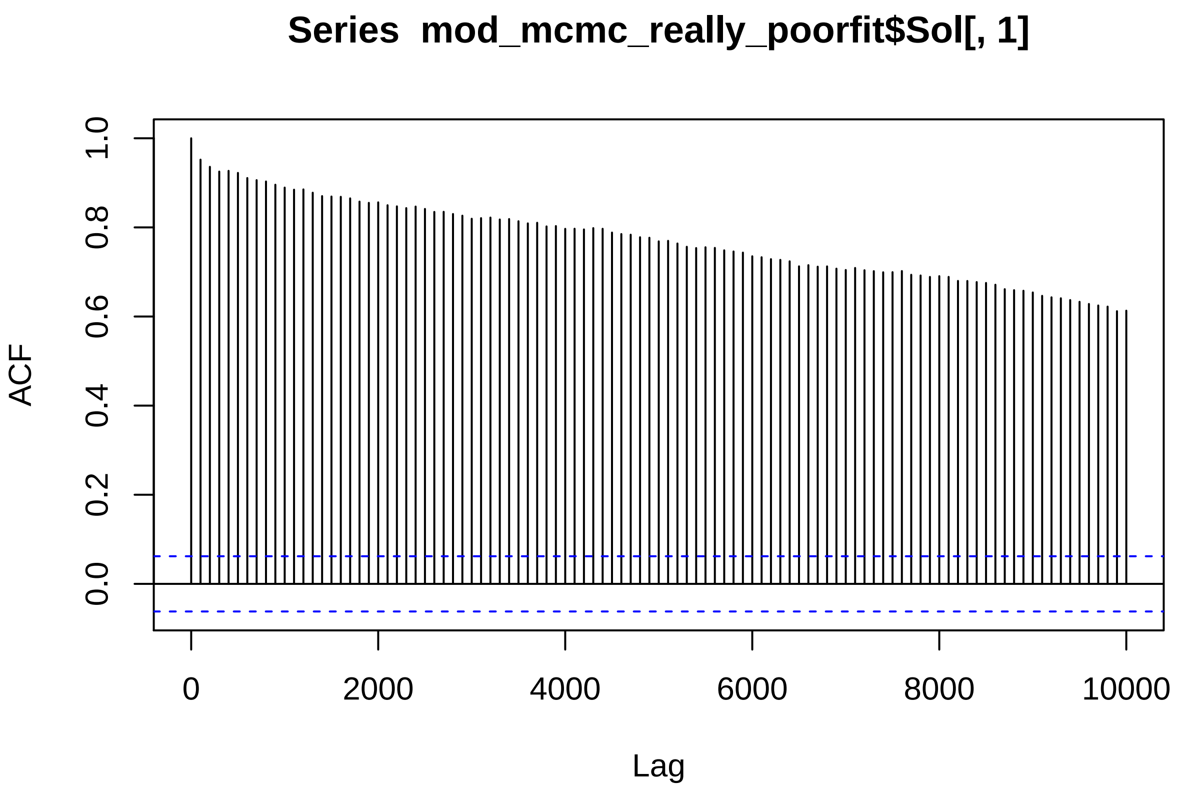
```
##          (Intercept)         mass  volantvolant
## Lag 0   1.000000000  1.000000000  1.000000e+00
## Lag 1  -0.002774028  0.010236940  5.350125e-05
## Lag 5  -0.002390219 -0.002114861 -3.025358e-03
## Lag 10  0.011090349  0.014800045  9.345110e-03
## Lag 50 -0.008335992 -0.001594042  1.770831e-02
```

```
autocorr.diag(mod_mcmc_good_fit$VCV)
```

```
##              genus      units
## Lag 0  1.000000000 1.00000000
## Lag 1  0.280535179 0.74713743
## Lag 5  0.008945061 0.24525374
## Lag 10 0.006685572 0.06789811
## Lag 50 0.003465391 0.02608561
```

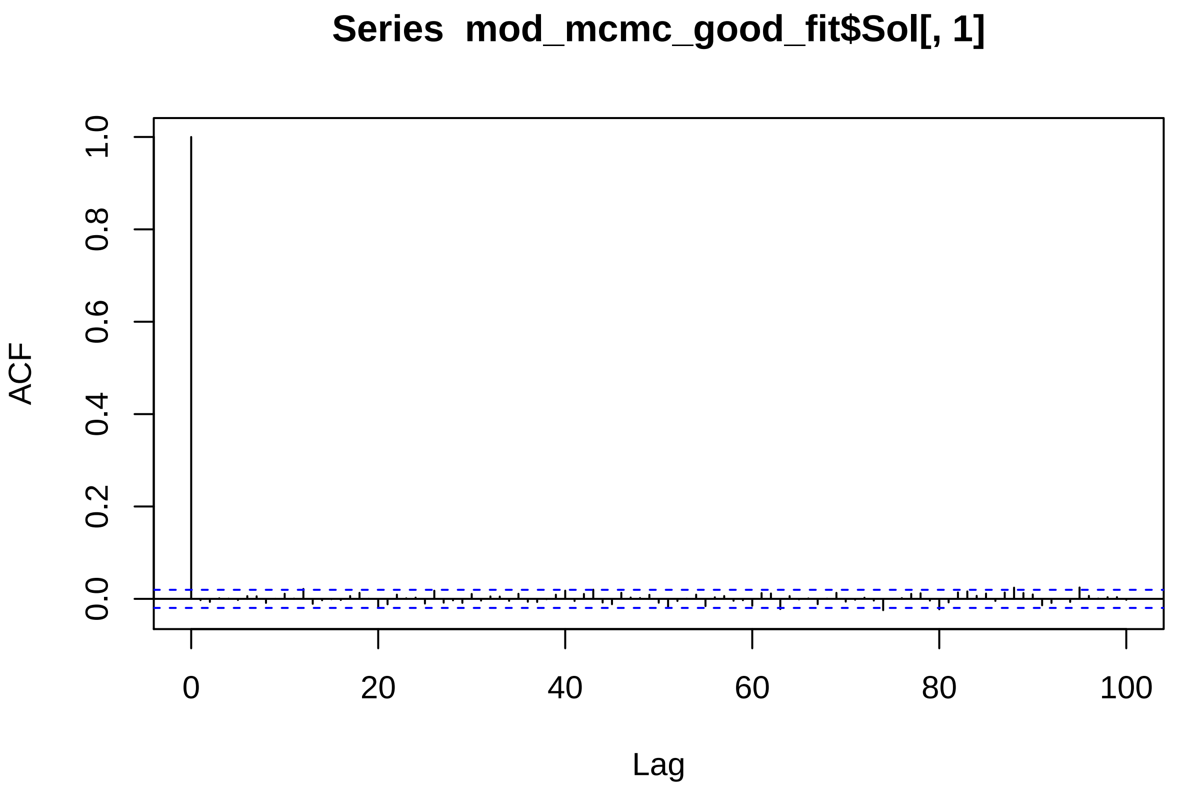
or we can look at autocorrelation plots for each of the traces. For example, let's check the auto-correlation in the intercept chain using the `acf` function

```
#acf plot for the first fixed estimate in our model (the intercept)
acf(mod_mcmc_really_poorfit$Sol[,1], lag.max =100)
```



However, in our other model the autocorrelation looks much better

```r
#acf plot for the first fixed estimate in our model (the intercept)
acf(mod_mcmc_good_fit$Sol[,1], lag.max =100)
```



For our intercept the auto-correlation plot looks good. However if there is bad auto-correlation one quick way to deal with this is to simply increase the thinning. While we don't need to in our case an example would be running something like

```
nitt2 <- 240000
burnin2 = 40000
thin2 = 100

mod_mcmc_long <- MCMCglmm(fixed = longevity ~ mass + volant,
                     family="gaussian",
                     data = lifespan_mammals,
                     nitt = nitt2,
                     burnin = burnin2,
                     thin = thin2,
                     prior = prior,
                     verbose=FALSE)
```

Noticed I also increased the number of iterations. One rough and ready rule that I like to use is to aim for an effective sample size of somewhere between 1000-2000 for all my estimates. The effective sample size is the number of samples in the posterior after the burnin, thinning and autocorrelation are accounted for. As the name suggests its a measure of the number of independent iterations in our chain. The summary of the model gives these so this is a very useful thing to watch out for.

```
summary(mod_mcmc_long)
```

```
##
##  Iterations = 40001:239901
##  Thinning interval  = 100
##  Sample size  = 2000
##
##  DIC: 270.113
##
##  R-structure:  ~units
##
##       post.mean l-95% CI u-95% CI eff.samp
## units    0.4295   0.3309   0.5341     2000
##
##  Location effects: longevity ~ mass + volant
##
##              post.mean l-95% CI u-95% CI eff.samp  pMCMC
## (Intercept)    0.02375 -0.16285  0.20487     2170  0.807
## mass           0.45597  0.33659  0.56471     2000 <5e-04 ***
## volantvolant   0.95466  0.43717  1.41834     2000 <5e-04 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

One last thing to check is that our MCMC chain has properly converged and that our estimate is not the result of some type of transient behaviour. That is if we run more than one MCMC chain does it arrive at the same value estimate for each of our terms. If not we may need to run our chains for longer or check that our model is behaving. To check this we will run a second model and see if it converges on the same estimates as our first model.

```
mod_mcmc_2 <- MCMCglmm(fixed = longevity ~ mass + volant,
                     family="gaussian",
                     data = lifespan_mammals,
                     nitt = nitt2,
                     burnin = burnin2,
                     thin = thin2,
                     prior = prior,
                     verbose=FALSE)
```

We can now check the convergence of the two chains using the Gelman and Rubin Multiple Sequence Diagnostic. This calculates the within-chain and between-chain variance of the chains and then gives a scale reduced factor, (for more see here. When this number is close to one (something below 1.1 is usually good) the chains are indistinguishable and hence can be considered to be converged.

```
#checking convergence for our fixed factors
gelman.diag(mcmc.list(mod_mcmc_long$Sol, mod_mcmc_2$Sol))
```

```
## Potential scale reduction factors:
##
##              Point est. Upper C.I.
## (Intercept)           1          1
## mass                  1          1
## volantvolant          1          1
##
## Multivariate psrf
##
## 1
```

```
#checking convergence for our random terms
gelman.diag(mcmc.list(mod_mcmc_long$VCV, mod_mcmc_2$VCV))
```

```
## Potential scale reduction factors:
##
##       Point est. Upper C.I.
## units          1       1.01
```

Once you are happy with the above model you have essentially got the basics of running a Bayesian mixed effects model. Building on the complexity involves adding more terms which are covered in good detail in the course notes and vignette.

Bonus materials

Non-normal response variables

Running a MCMCglmm seems like a lot of work to get the same result so why bother. One reason is that you are fundamentally team Bayesian and like the philosophy of it all. Another is that MCMCglmm can give you a lot of flexibility in your models. For example, we can run models with different types of response variables such as a categorical response term by simply changing the family input to "categorical".

So if we wanted to run a model with volant as a response variable we could run something like

```
mod_mcmc_cat <- MCMCglmm(fixed = volant ~ mass,
                         rcov=~ units,
                         random= ~ genus,
                         family="categorical",
                         data = lifespan_mammals,
                         nitt = nitt2,
                         burnin = burnin2,
                         thin = thin2,
                         prior = prior_mixed,
                         verbose=FALSE)
summary(mod_mcmc_cat)
```

```
##
##  Iterations = 40001:239901
##  Thinning interval  = 100
##  Sample size  = 2000
##
##  DIC: 0.3829699
##
##  G-structure:  ~genus
##
##       post.mean  l-95% CI u-95% CI eff.samp
## genus     71432 0.0003943   128007    13.16
##
##  R-structure:  ~units
##
##       post.mean l-95% CI u-95% CI eff.samp
## units      8454 0.001129    57437    11.93
##
##  Location effects: volant ~ mass
##
##             post.mean l-95% CI u-95% CI eff.samp  pMCMC
## (Intercept)    -358.7   -486.8   -218.2   14.255 <5e-04 ***
## mass           -296.7   -397.5   -164.7    5.825 <5e-04 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

*Unfortunately this model runs terrible and does not converge. However, this does still demonstrate the ability to run such a model.

This flexibility is particularly useful as random effects can also easily be incorporated in such models unlike frequentest approaches that require more complex maths to solve over more complex likelihood spaces where Bayesian approaches can "find" such estimates using computational power to search such complex spaces. To get a full list of the families `MCMCglmm` can deal with check out `??MCMCglmm`

Multiple response in a MCMCglmm

Another advantage of using MCMCglmm is that you can build multiple response models. These might be useful when you are interested in the effects on two variables simultaneously hence avoiding the need to treat one as an explanatory variable. As before we need to set some priors.

```
prior_mul<-list(R = list(V = diag(2)/2, nu=0.002), G = list(G1=list(V = diag(2)/2,nu = 0.002)))
```

Since we want to fit separate parameters for each response variable (i.e we want an slope/intercept for both longevity and volancy) we use the diag function again.

Now we are set to run a multiple response model. We need to use the `us()` and `idh()` aspects of the model. The notation `us()` specifies parameters for all variances and co-variances, giving the same random-effects parameters that lme4 would give us by default. The idh() tells MCMCglmm to estimate parameters for just the variances for the random term but not the co-variances.

```
mod_mul_r <- MCMCglmm(cbind(longevity,volant) ~ trait:mass,
                      rcov=~us(trait):units,
                      random= ~ idh(trait):genus,
                      family= c("gaussian","catagorical"),
                      data = lifespan_mammals,
                      nitt = nitt2,
                      burnin = burnin2,
                      thin = thin2,
                      prior = prior_mul,
                      verbose=FALSE)
summary(mod_mul_r)
```

```
##
##  Iterations = 40001:239901
##  Thinning interval  = 100
##  Sample size  = 2000
##
##  DIC: -631.1706
##
##  G-structure:  ~idh(trait):genus
##
##                      post.mean l-95% CI u-95% CI eff.samp
## traitlongevity.genus    0.4269   0.2953   0.5719   2000.0
## traitvolant.2.genus   105.7171  21.0301 197.6253     10.2
##
##  R-structure:  ~us(trait):units
##
##                                    post.mean   l-95% CI u-95% CI eff.samp
## traitlongevity:traitlongevity.units   0.06828  0.0396659   0.1043  1646.31
## traitvolant.2:traitlongevity.units    0.25659 -0.5000067   1.4006    39.93
## traitlongevity:traitvolant.2.units    0.25659 -0.5000067   1.4006    39.93
## traitvolant.2:traitvolant.2.units     4.15271  0.0009462  22.5801    63.12
##
##  Location effects: cbind(longevity, volant) ~ trait:mass
##
##                      post.mean   l-95% CI   u-95% CI eff.samp  pMCMC
## (Intercept)           0.165900  -0.008579   0.347760  656.038  0.071 .
## traitlongevity:mass   0.380638   0.267292   0.489761 1331.582 <5e-04 ***
## traitvolant.2:mass  -11.071947 -15.231235  -5.681207    6.117 <5e-04 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Like before we get our fixed effects now under the Location effects section, however now we get an estimate for both mass against both longevity and volancy (again the results of this model have not converged at all so effect sizes are not to be trusted).

Parameter expanded priors

I noted above that one problem that may occur is a difficulty getting rid of auto-correlation. One reason this may happen is that when values for variance terms are close to zero chains can become "stuck" affecting the mixing of the model. This would be seen as trace plots that show poor mixing, and don't look like hairy caterpillars. One potential solution is to use parameter expanded priors. These work by splitting the prior into two components, one part of the prior deals with values very close to zero and the other part of the prior deals with all other values. This split allows the chain to be bounced out of very low values and help with the chain mixing.

To set one, we include an `alpha.mu` and an `alpha.V` which essentially set the extra part of the prior. To set this part of the prior we put `alpha.mu` as a vector of zeros equal to the number of random effects and set `alpha.V` as a nxn matrix with very large numbers were `n` is the number of random effects. In this case we only have one random effect so its easy 1 zero and 1 large number.

```
prior_exp<-list(R = list(V = 1, nu=0.002), G = list(G1=list(V = 1,n = 1, alpha.mu=rep(0,1), alpha.V= di
```

We can use this in our categorical model for example.

```
mod_mcmc_cat_exp <- MCMCglmm(fixed = volant ~ mass,
                     random= ~ genus,
                     family="categorical",
                     data = lifespan_mammals,
                     nitt = nitt2,
                     burnin = burnin2,
                     thin = thin2,
                     prior = prior_exp,
                     verbose=FALSE)
summary(mod_mcmc_cat)
```

```
##
##  Iterations = 40001:239901
##  Thinning interval  = 100
##  Sample size  = 2000
##
##  DIC: 0.3829699
##
##  G-structure:  ~genus
##
##       post.mean  l-95% CI u-95% CI eff.samp
## genus     71432 0.0003943   128007    13.16
##
##  R-structure:  ~units
##
##       post.mean l-95% CI u-95% CI eff.samp
## units      8454 0.001129    57437    11.93
##
##  Location effects: volant ~ mass
##
##             post.mean l-95% CI u-95% CI eff.samp  pMCMC
## (Intercept)    -358.7   -486.8   -218.2   14.255 <5e-04 ***
## mass           -296.7   -397.5   -164.7    5.825 <5e-04 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Again this model would need to be run for much longer and even with the parameter expanded prior it may have problems converging.