MCMCglmm in comparative methods and Data Imputation

Kevin Healy

14 July 2016

This is a short example of running some simple phylogenetic comparative analysis using both pgls and MCMCglmm and also using MCMCglmm to impute data.

Before jumping back into MCMCglmm we will use the caper package to import and manage our phylogentic data: caper package see here For more on MCMCglmm see the course notes or vignettes.

Installation

First we need to install some packages including the ape and caper packages that run the pgls models and the MCMCglmm package to run the Bayesian version of the phylogenetic comparative analysis.

```
if(!require(ape)) install.packages("ape")
if(!require(caper)) install.packages("caper")
if(!require(MCMCglmm)) install.packages("MCMCglmm")
```

We will also install from GitHub the `MulTree` package which is still under development (so watch out for BUGS) but contains some handy data and also will allow us to use MCMCglmm to include the error associated with building phylogenies within our analysis later in the session. We used this data yesterday so you should be familiar with it. TO get it again we need to go back to GitHub To do so we need to get them from GitHub and so we need to run.

```
#if(!require(devtools)) install.packages("devtools")
library(devtools)
install_github("TGuillerme/mulTree", ref = "master")
```

Now we load up the packages, and we are good to go.

```
library(ape)
library(caper)
library(MCMCglmm)
library(mulTree)
```

Data

We will use some handy data that is part of a `MulTree` package that contains some trees and data that are ready to go.

```
data(lifespan)
```

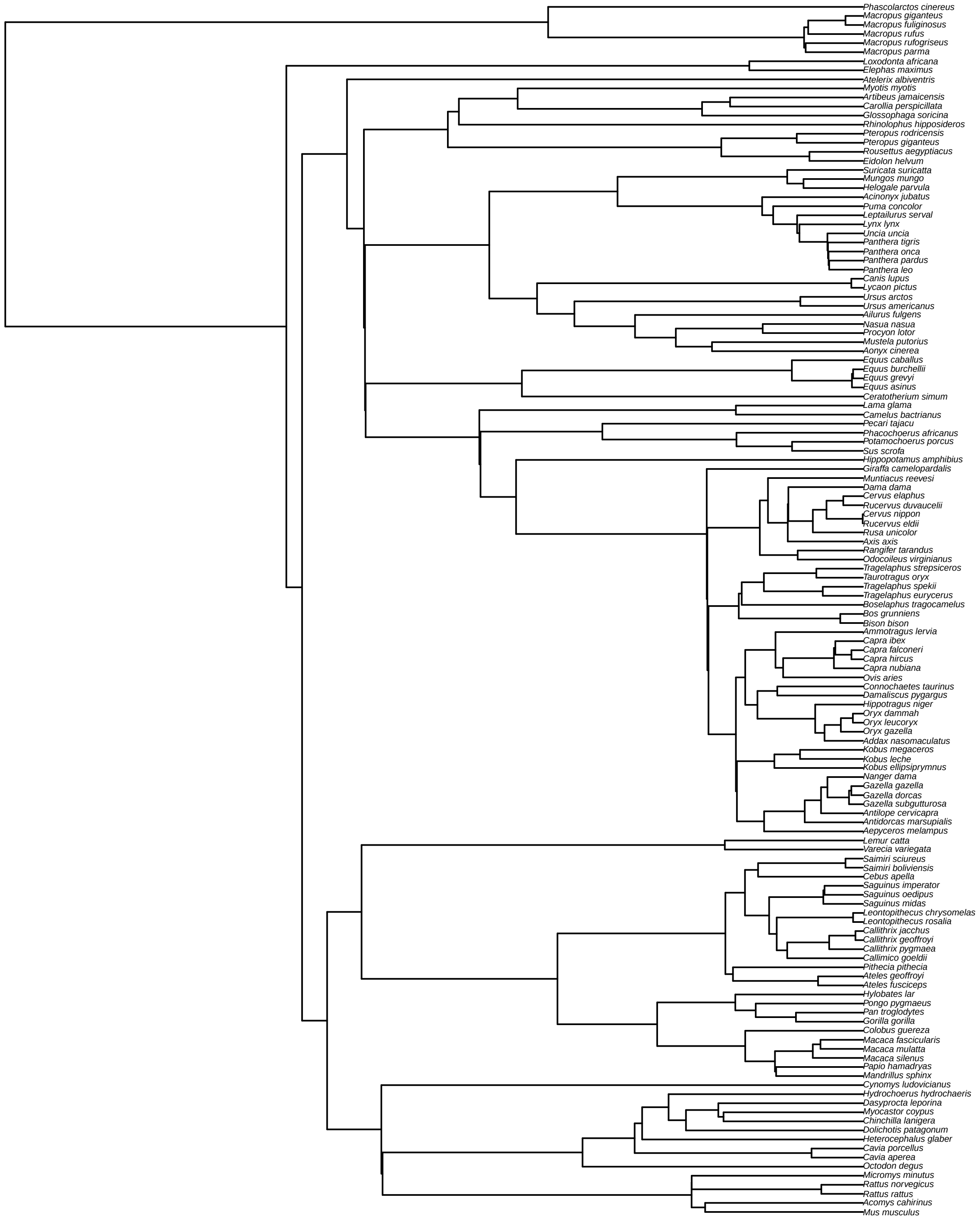
This data file contains a subset of the data used in an analysis on the role of flying (volant) in the evolution of maximum lifespan in birds and mammals Link to paper. Note that these data have been log transformed, mean centered and expressed in units of standard deviation. The original lifespan data were taken from the Anage database.

```
#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
```

We will use a phylogeny of mammals constructed in `Kuhn et al 2011`, were they produce 10,000 trees with each individual tree comprising one resolution of the polytomies of a previously published supertree. For now we will just use the first tree in the list, later we will return to see how we might include a range of trees.

```
# The 10Ktrees from Khun et al (2011) gives a set of trees to represent different polytomies.
# For now let just take one.
mammal_tree <- trees_mammalia[[1]]
plot(mammal_tree, cex = 0.3)
```



```
#the number of species
Ntip(mammal_tree)
```

```
## [1] 134
```

```
#we can also check that its ultrametric
is.ultrametric(mammal_tree)
```

```
## [1] TRUE
```

Lets run some models

Lets 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",]

###lets define our fixed factors
formula_a <- longevity ~ mass + volant

#### and run a simple glm.libr
glm_mod <- glm(formula = formula_a, family = "gaussian", data = lifespan_mammals)
summary(glm_mod)
```

```
##
## Call:
## glm(formula = formula_a, 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 ran this model yesterday but just to recap so far we are assuming that each data point is independent. We assume that even if two species are closely related to each other that their lifespans are in no way related. We know however that the traits of two very closely related species are likely to be related. For example, a close relative of a large animal, like a whale, would also be expected to be large, especially if the two species split in recent evolutionary time. We accounted for this yesterday to some degree using a random effect of genus but that doesnt really cut the mustard as we have a lot more information we could include. A simple way to do this is using phylogentic information. To include the non-independent nature of our data we will first turn to Phylogenetic generalized linear models (PGLS).

PGLS

In phylogenetic comparative methods we try to deal with this non-independence by using the structure of a phylogeny to weight the error term so that our model fit is no longer blind to the non-independence of our data.

The first step is to make sure that our data and phylogeny match up correctly. We will use the comparative.data function from the caper package to make sure that the species in the dataset are matched up to the species in the phylogeny. To do so we need to give the phylogeny (phy); the dataset and the name of the column in the dataset with the species names (names.col). As we want to calculated a variance-covariance matrix of the phylogeny we set vcv = true.

```
comp_data <- comparative.data(phy = mammal_tree,
                              data =lifespan_volant,
                              names.col = species,
                              vcv=TRUE)

head(comp_data$data)
```

```
##                        class longevity       mass    volant
## Mus_musculus        Mammalia -2.056791 -1.2680281 nonvolant
## Acomys_cahirinus    Mammalia -1.477934 -0.9776073 nonvolant
## Rattus_rattus       Mammalia -1.984124 -0.4313382 nonvolant
## Rattus_norvegicus   Mammalia -2.133185 -0.2824073 nonvolant
## Micromys_minutus    Mammalia -2.133185 -1.7193276 nonvolant
## Octodon_degus       Mammalia -0.190961 -0.3721030 nonvolant
```

```
##notice in the comp_data$data that there are now no birds
###these have been dropped
head(comp_data$dropped)
```

```
## $tips
## character(0)
##
## $unmatched.rows
##  [1] "Buteo_jamaicensis"        "Gyps_fulvus"
##  [3] "Haliaeetus_leucocephalus" "Parabuteo_unicinctus"
##  [5] "Anas_acuta"               "Anser_anser"
##  [7] "Aythya_marila"            "Branta_bernicla"
##  [9] "Cygnus_olor"              "Dendrocygna_viduata"
## [11] "Uria_aalge"               "Leptoptilos_crumeniferus"
## [13] "Colius_striatus"          "Caloenas_nicobarica"
## [15] "Columba_livia"            "Dacelo_novaeguineae"
## [17] "Falco_tinnunculus"        "Gallus_gallus"
## [19] "Pavo_cristatus"           "Phasianus_colchicus"
## [21] "Grus_virgo"               "Balearica_regulorum"
## [23] "Cinclus_cinclus"          "Petrochelidon_pyrrhonota"
## [25] "Riparia_riparia"          "Ficedula_hypoleuca"
## [27] "Sericornis_frontalis"     "Parus_major"
## [29] "Sturnus_vulgaris"         "Zosterops_lateralis"
## [31] "Pelecanus_crispus"        "Pelecanus_onocrotalus"
## [33] "Geronticus_eremita"       "Threskiornis_aethiopicus"
## [35] "Phoenicopterus_roseus"    "Phoebastria_immutabilis"
## [37] "Phoebastria_nigripes"    "Amazona_aestiva"
## [39] "Ara_ararauna"            "Ara_macao"
## [41] "Ara_militaris"           "Cacatua_galerita"
## [43] "Cacatua_moluccensis"     "Cyanoliseus_patagonus"
## [45] "Eclectus_roratus"        "Cacatua_roseicapilla"
## [47] "Melopsittacus_undulatus" "Myiopsitta_monachus"
## [49] "Nymphicus_hollandicus"   "Poicephalus_senegalus"
## [51] "Psittacula_krameri"      "Psittacus_erithacus"
## [53] "Aptenodytes_patagonicus" "Spheniscus_demersus"
## [55] "Bubo_bubo"               "Dromaius_novaehollandiae"
## [57] "Struthio_camelus"        "Sula_dactylatra"
```

We can now run some models, first lets run two models with lambda set to 1 and something close to 0.

```
#we have the formula and the comparative.data
#object comp_data which contains but the phylogeny and the data.
#Lets set the lambda in this case to 1.

pgls_l1 <- pgls(formula = formula_a, data = comp_data, lambda = c(1))
pgls_l0 <- pgls(formula = formula_a, data = comp_data, lambda = c(0.01))
summary(pgls_l1)
```

```
##
## Call:
## pgls(formula = formula_a, data = comp_data, lambda = c(1))
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -0.29647 -0.08937 -0.02065  0.06000  0.67568
##
## Branch length transformations:
##
## kappa  [Fix]  : 1.000
## lambda [Fix]  : 1.000
## delta  [Fix]  : 1.000
##
## Coefficients:
##               Estimate Std. Error t value  Pr(>|t|)
## (Intercept)   1.053484   0.792826  1.3288    0.1862
## mass         -0.433933   0.078131 -5.5539 1.495e-07 ***
## volantvolant -1.059141   0.812158 -1.3041    0.1945
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.1252 on 131 degrees of freedom
## Multiple R-squared: 0.1906,  Adjusted R-squared: 0.1782
## F-statistic: 15.42 on 2 and 131 DF,  p-value: 9.663e-07
```

```
summary(pgls_l0)
```

```
##
## Call:
## pgls(formula = formula_a, data = comp_data, lambda = c(0.01))
```

```
##
## Residuals:
##       Min        1Q    Median        3Q       Max
## -0.226378 -0.021377  0.002155  0.017977  0.290707
##
## Branch length transformations:
##
## kappa  [Fix]  : 1.000
## lambda [Fix]  : 0.010
## delta  [Fix]  : 1.000
##
## Coefficients:
##               Estimate Std. Error t value  Pr(>|t|)
## (Intercept)  0.0011739  0.0991748  0.0118 0.9905740
## mass         0.4760918  0.0586018  8.1242 2.931e-13 ***
## volantvolant 0.9950322  0.2533605  3.9273 0.0001382 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.05256 on 131 degrees of freedom
## Multiple R-squared: 0.335,   Adjusted R-squared: 0.3249
## F-statistic:    33 on 2 and 131 DF,  p-value: 2.474e-12
```

The outputs looks similar to our glm with the estimates of the Coefficients, Adjusted R-squared etc. However Next lets run a model were lambda is no longer fixed. We can do this by specifying lambda = "ML" which tells the model to estimate it using maximum likelihood.

```r
#Finally we also need to set the lambda in this case to ML.
#This means the we will using Maximum Likelihood
#to calculate the lambda.
pgls_mod <- pgls(formula = formula_a, data = comp_data, lambda = "ML")
summary(pgls_mod)
```

```
##
## Call:
## pgls(formula = formula_a, data = comp_data, lambda = "ML")
##
## Residuals:
##       Min        1Q    Median        3Q       Max
## -0.194491 -0.038027  0.004578  0.031555  0.151222
##
## Branch length transformations:
##
## kappa  [Fix]  : 1.000
## lambda [ ML]  : 0.933
##    lower bound : 0.000, p = < 2.22e-16
##    upper bound : 1.000, p = < 2.22e-16
##    95.0% CI    : (0.866, 0.968)
## delta  [Fix]  : 1.000
##
## Coefficients:
##              Estimate Std. Error t value  Pr(>|t|)
## (Intercept)  0.076286   0.394227  0.1935   0.84686
## mass         0.416752   0.070962  5.8729 3.334e-08 ***
## volantvolant 0.988252   0.427932  2.3094   0.02249 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.06309 on 131 degrees of freedom
## Multiple R-squared: 0.2084,  Adjusted R-squared: 0.1963
## F-statistic: 17.25 on 2 and 131 DF,  p-value: 2.246e-07
```

Now under Branch length transformations we also now get the estimated branch transformation under maximum likelihood. As we are only interested in fitting only lambda for now the other types of transformations, (kappa and delta), are held fixed.

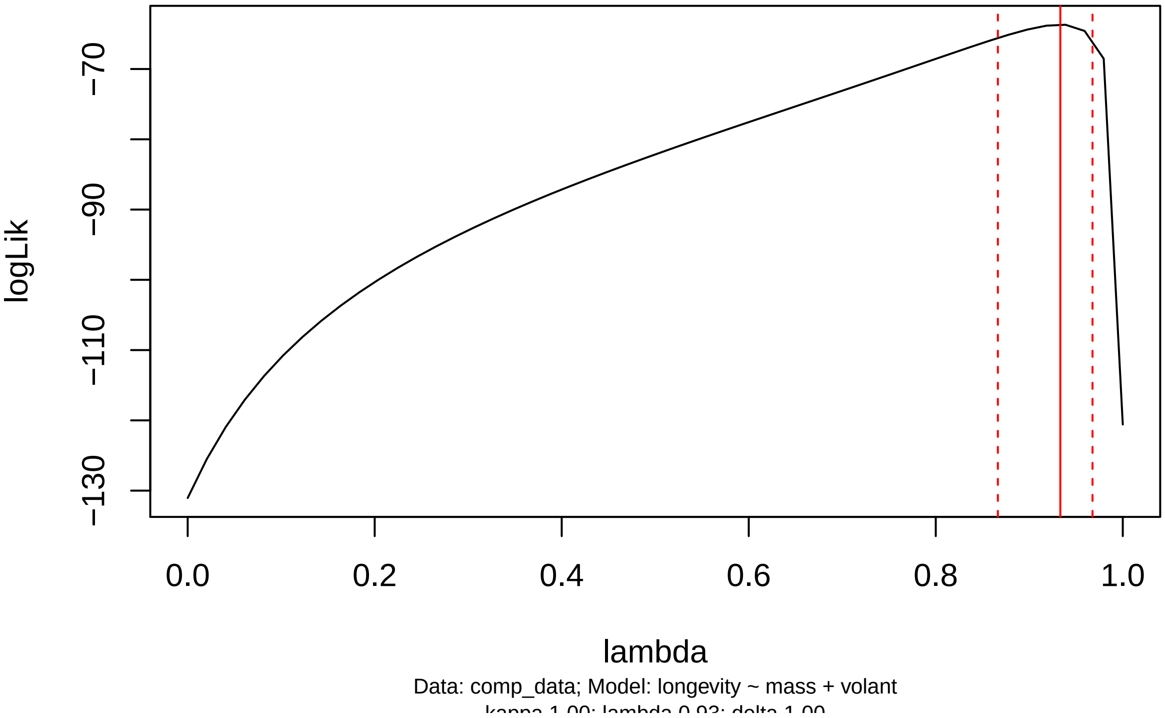
Lambda here estimated as

```
pgls_mod$param["lambda"]
```

```
##    lambda
## 0.9331958
```

As it is close to 1 the traits in this model are correlated under Brownian motion. If our value was 0 it would indicate that our data points are essentially independent. We can then go a check various elements of the model such as the likelihood profile for lambda.

```
mod_profile <- pgls.profile(pgls_mod)
plot(mod_profile)
```



which looks good as the profile shows a nice clear peak.

We would also then go ahead and check our residuals etc but for now we will assume everything is good and move onto running a similar model using MCMCglmm

MCMCglmm

So far we have fitted a very simple glm and a pgls model that included phylogeny to account for non-independence. Now we will use a Bayesian approach were we include phylogeny as a random term using the animal model in the MCMCglmm package.

As we are using a Bayesian approach we will first set up the priors. In most cases we want to use a non-informative prior that doesn't influence the estimated posterior distribution. For the random effect prior we will use 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 descripted as V = 1 and nu = 0.002. (For more on priors for the animal model see course notes)

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

We describe our prior as above for the random (G) and residual variances (R) each of them as a list, which we will in turn put within a list. If we wanted to include more random terms we would include a G2, G3 etc for each additional random term within the G list. We could also specify priors for the fixed terms using B, however MCMCglmm will automatically do that for us and as it usually does a good job at it we will ignore it here.

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 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 lets 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.

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 need to set up the data. We have already cleaned and matched up our data earlier using the comparative.data function but we need to now add an extra column into our dataset called "animal" which contains the species matched between the tree and the data.

```
#Matched data
mcmc_data <- comp_data$data
#As MCMCglmm requires a colume named animal for it to identify it
#as a phylo model we include an extra colume with the species names in it.
mcmc_data <- cbind(animal = rownames(mcmc_data), mcmc_data)
mcmc_tree <- comp_data$phy
```

MCMCglmm reserves the random variable "animal" to call a model that includes the phylogeny as an additive genetic effect. If we name it something else, like say "species", MCMCglmm will either throw an error looking for "animal", or if we do not provide a phylogeny under pedigree it will run "species" like a standard random term. Now we can run the model.

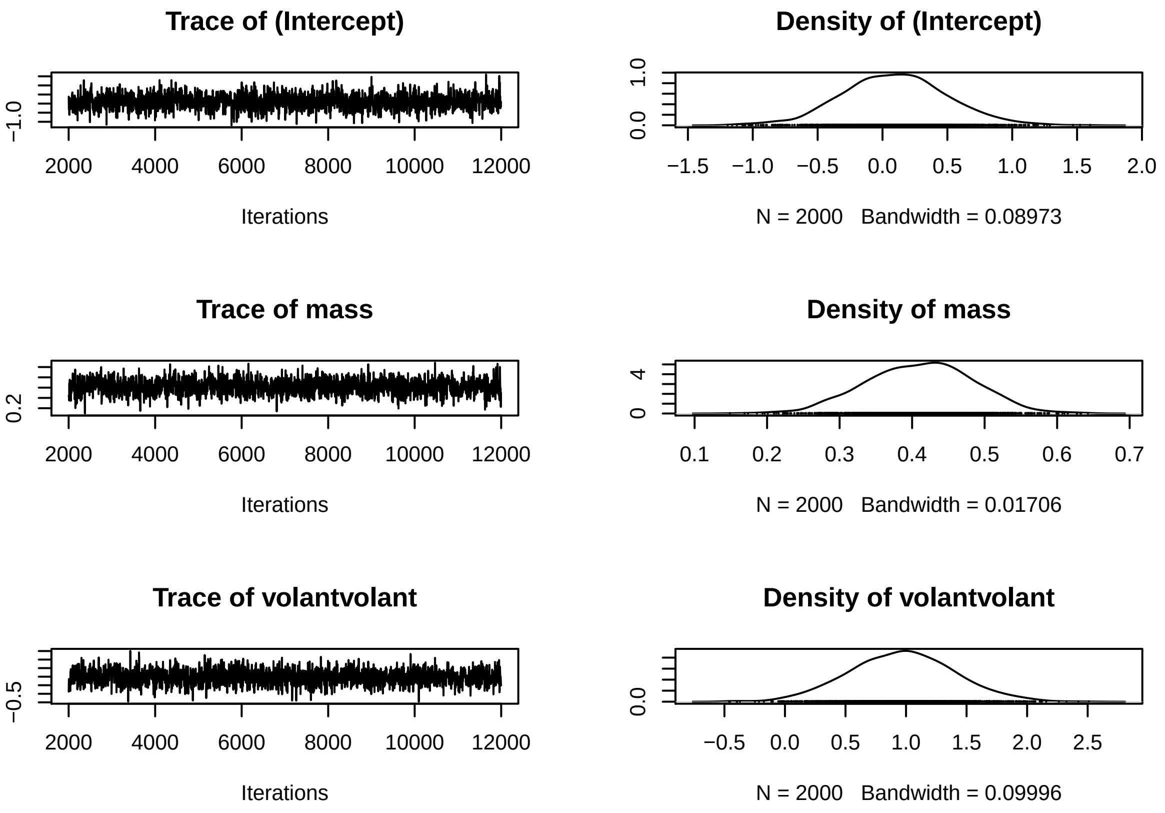
```r
mod_mcmc <- MCMCglmm(fixed = formula_a,
                     random= ~ animal,
                     family="gaussian",
                     pedigree = mcmc_tree,
                     data = mcmc_data,
                     nitt = nitt,
                     burnin = burnin,
                     thin = 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
```

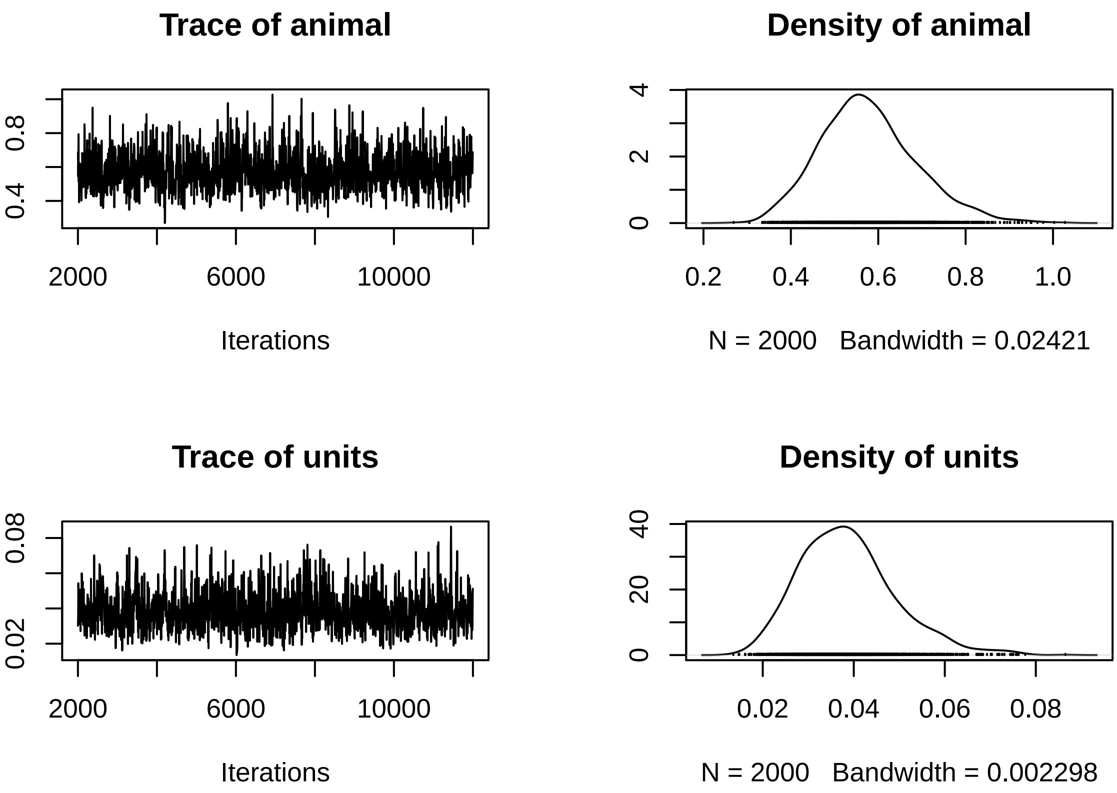
As the model runs we see the iterations print out. These chains can take some time to run, depending on the model, however, since we only ran our chains for 12000 iterations it doesnt take long here.

Before we even look at our model we need to check if the model ran appropriately. We can do this by visually inspecting the chains to make sure there has been no unruly behaviour! We can extract the full chains using model$Sol for the fixed effects and model$VCV for the random effect variances. So Sol[,1] will give you the first fixed term, in this case the intercept, and VCV[,1] will give you the first random term, which is “animal” and so on. As our model is an mcmc object when we use the plot function we get a trace plot.

```r
plot(mod_mcmc$Sol)
```

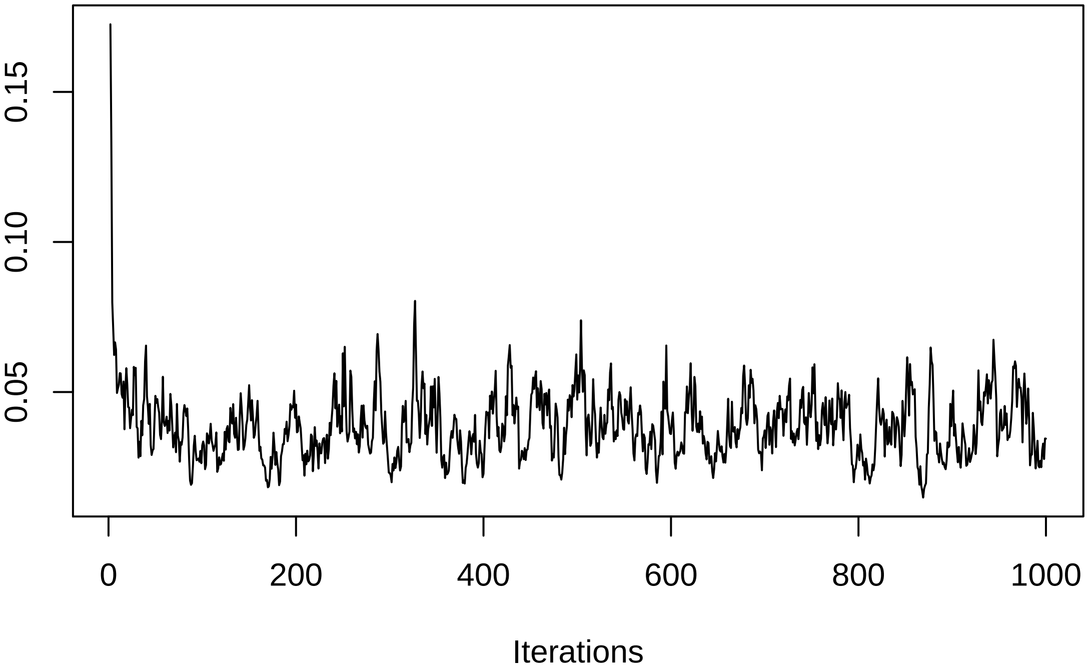


```
plot(mod_mcmc$VCV)
```



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. What we want to see in these trace plots is "hairy caterpillars" (not my phrase!). That is a trace with no obvious trend that is bouncing around some stable point.

What we don't want to see in the trace plots can be demonstrated if we only run our model over a very short chain (itt == 1000). Notice that without a burnin the start of trace is well outside the area that the chain will converges towards.



So in our longer run model everything looks good visually, however we also want to check the level of autocorrelation 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$Sol)
```

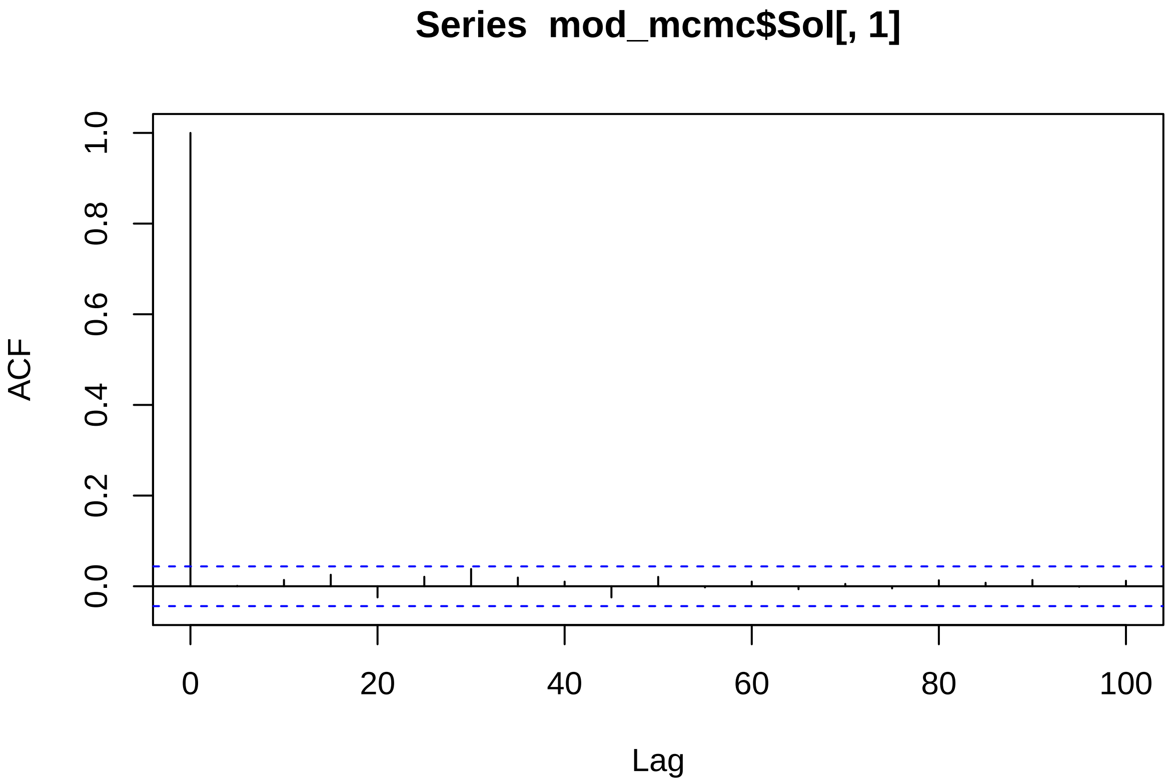
```
##           (Intercept)          mass volantvolant
## Lag 0    1.0000000000  1.000000000  1.000000000
## Lag 5    0.0008321781  0.047677210 -0.009501558
## Lag 25   0.0208996890  0.004797944  0.050850781
## Lag 50   0.0206502414 -0.004917536 -0.005027047
## Lag 250 -0.0131307643  0.005060249 -0.013713527
```

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

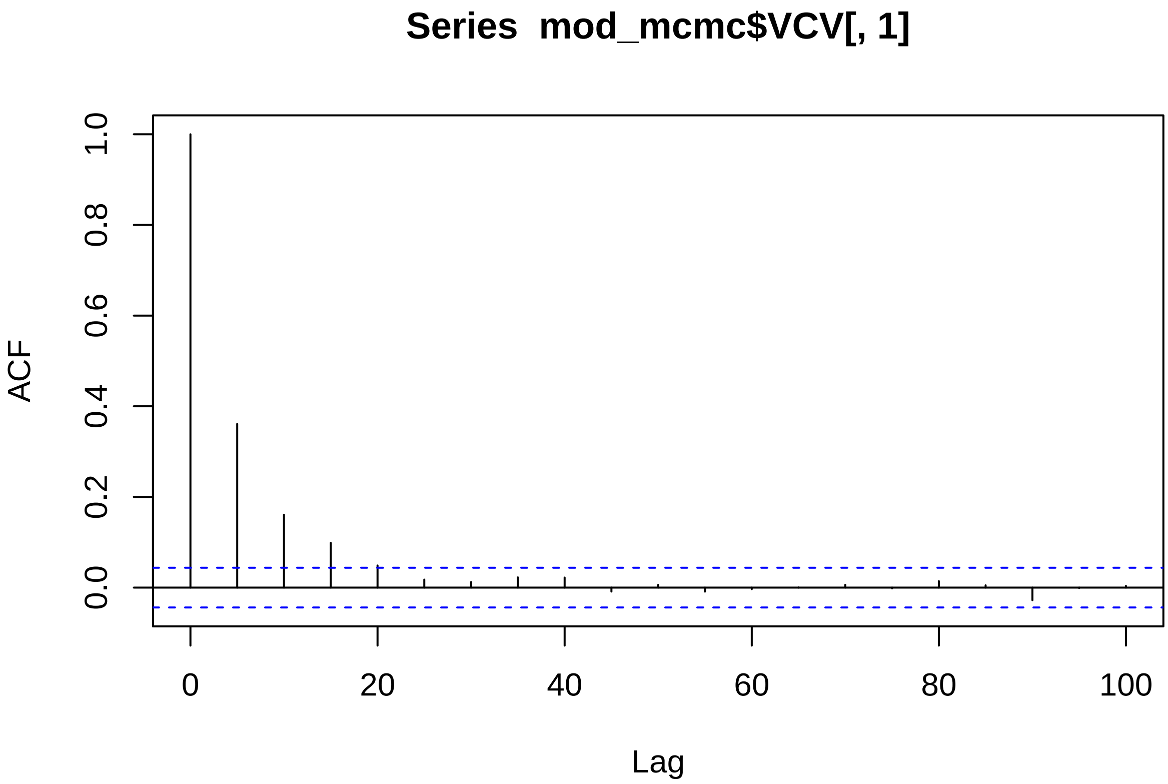
```
##              animal         units
## Lag 0   1.000000000  1.000000000
## Lag 5   0.361191859  0.365749950
## Lag 25  0.017678126  0.059982029
## Lag 50  0.005988061  0.004416807
## Lag 250 0.010178656 -0.004016805
```

or we can look at autocorrelation plots for each of the traces, we'll look at just one using the acf function here.

```
#acf plot for the first fixed estimate in our model (the intercept)
acf(mod_mcmc$Sol[,1], lag.max =20)
```



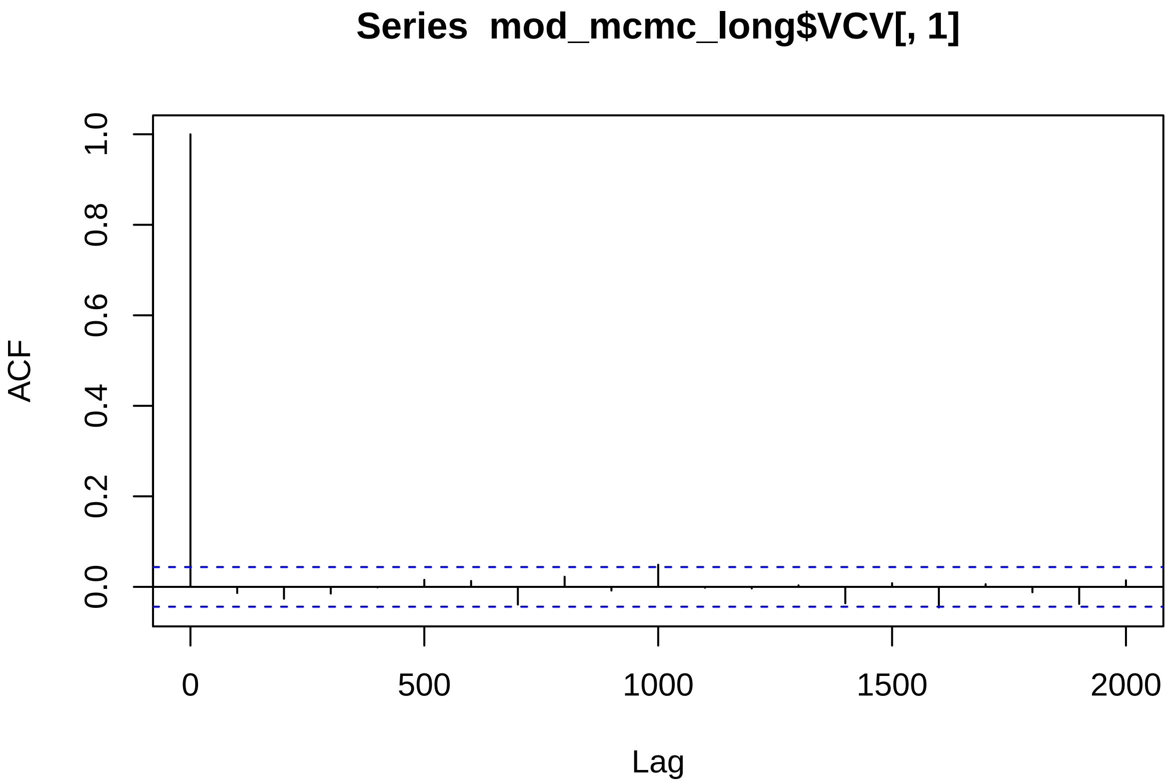
```
#acf plot for the first random term in our model (the animal term)
acf(mod_mcmc$VCV[,1], lag.max =20)
```



For our intercept the autocorrelation plot looks good, however the animal term still shows some autocorrelation. One quick way to deal with this is to simply increase the thinning.

```
nitt2 <- 240000
burnin2 = 40000
thin2 = 100
mod_mcmc_long <- MCMCglmm(fixed = formula_a,
                     random= ~ animal,
                     family="gaussian",
                     pedigree = mcmc_tree,
                     data = mcmc_data,
                     nitt = nitt2,
                     burnin = burnin2,
                     thin = thin2,
                     prior = prior,
                     verbose=FALSE)

acf(mod_mcmc_long$VCV[,1], lag.max =20)
```



That looks better now. 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 my chains, which is the number of iterations used in the posterior after the burnin, thinning and accounting for autocorrelation, somewhere between 1000-2000.

```
#acf plot for the first fixed estimate in our model (the intercept)
effectiveSize(mod_mcmc_long$Sol)
```

```
##  (Intercept)         mass volantvolant
##     2133.216     2000.000     2000.000
```

```
effectiveSize(mod_mcmc_long$VCV)
```

```
## animal  units
##   2000   2000
```

*One thing to note is that while thinning might help autocorrelation it wont solve it and you might have to use parameter expanded priors. These are priors that help weight the chain away from zero, a common problem when variance is low or with certain phylogenetic structures. They work by splitting the prior into 2 components with one component weighing the chain away from zero.

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 transitional behaviour. That is have our chains "found" the optimum or do we need to let them run longer before they settle around some estimate. 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 = formula_a,
                       random= ~ animal,
                       family="gaussian",
                       pedigree = mcmc_tree,
                       data = mcmc_data,
                       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 (say below 1.1) 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.
## animal          1          1
## units           1          1
##
## Multivariate psrf
##
## 1
```

Since everything looks good, we will finally look at the results of our model.

```
summary(mod_mcmc_long)
```

```
##
##  Iterations = 40001:239901
##  Thinning interval  = 100
##  Sample size  = 2000
##
##  DIC: 23.47866
##
```

```
##  G-structure:  ~animal
##
##        post.mean l-95% CI u-95% CI eff.samp
## animal    0.5708   0.3496   0.7793     2000
##
##  R-structure:  ~units
##
##       post.mean l-95% CI u-95% CI eff.samp
## units   0.03911  0.02022  0.06217     2000
##
##  Location effects: longevity ~ mass + volant
##
##              post.mean l-95% CI u-95% CI eff.samp  pMCMC
## (Intercept)    0.07452 -0.71047  0.82013     2133  0.846
## mass           0.41071  0.26031  0.56008     2000 <5e-04 ***
## volantvolant   0.97075  0.15043  1.82008     2000  0.022 *
## ---
## 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 (Again notice the similarity to our pgls model). Each parameter has a measure of the effect size using the post.mean and a lower and higher 95% credible interval (CI). These are simply calculated from the posterior distributions we looked at in the above plots, so if you would rather calculated the median instead of using the mean we can simple use

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

```
## [1] 0.07076983
```

We also have the effective sample size (eff.samp) and the pMCMC which calculated as two times the probability that the estimate is either > or < 0, using which ever one is smaller. However since our data has been mean centred and expressed in units of standard deviation we can look at what proportion of our posterior is on either side of zero.

For the random terms we have the posterior distribution for our G-structure which includes or phylogenetic effect and the R-structure which is our residual variation.

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 better.

Finally, we can also calculate the H^2 which is comparable to pagels lambda as

```
H <- (var(mod_mcmc_long$VCV[,"animal"]))/
      (var(mod_mcmc_long$VCV[,"animal"])
       + var(mod_mcmc_long$VCV[,"units"]))
H
```

```
## [1] 0.9900515
```

Before moving on to the next section try running the above analysis subsetted for birds as opposed to mammals.

```
#aves tree from Jetz et al 2012
aves_tree <- trees_aves[[1]]
```

Extending MCMCglmm: including multiple trees

So far we have run our analysis over one single phylogeny. However we know phylogenies do not exist without uncertainty. For example, we only used a single tree from 10,000 in Kuhn et al (2011) and other phylogenies are now starting to be given as distributions such as the `Jetz et al (2012) bird phylogeny` One of the nice features about using MCMCglmm is that as the output is a posterior distribution we can simple run multiple models, one for each tree, and combine the output. This is starting to become more common and is a nice way to include the uncertainty relating to the phylogeny itself.

As an example of doing this we will use some Multree code (which is still in development at the moment and hence not on CRAN yet) that makes running these analysis easier for us.

For fun lets run a model over both birds and mammal using a subset of 2 trees from both Kuhn et al 2011 and Jetz 2012.

```
trees_aves
```

```
## 2 phylogenetic trees
```

```
trees_mammalia
```

```
## 2 phylogenetic trees
```

We need to graft these different phylogenies together, in this case we will use a root age of 250mya. If we only wanted one combined tree we would set sample = 1

```
combined_trees <- tree.bind(trees_mammalia, trees_aves, sample = 2, root.age = 250)
```

We will use the same data as before but this time we will keep the birds in it

```
data(lifespan)
###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
```

```
##lets package all the data up into one mulTree object
mulTree_data <- as.mulTree(data = lifespan_volant, tree = combined_trees,
taxa = "species")
```

We need to set up our parameters as before.

```
mul_formula <- longevity ~ mass + volant
## The MCMC parameters (iterations, thining, burnin)
mul_parameters <- c(nitt2, thin2, burnin2)
## The MCMCglmm priors
mul_priors <- list(R = list(V = 1, nu = 0.002),
G = list(G1 = list(V = 1, nu = 0.002)))
```

As running multiple mcmcglmm models can quickly cause issues with memory storage in R (100 trees would require at least 200 chains in order to test for convergence) MulTree exports each set of chains to you working directory only reading them back in when required. So make sure you are happy with wherever you are sending your models.

```
getwd()
```

```
## [1] "/Users/kevinhealy/Desktop/Cork teaching"
```

If we are all happy with that we can finally send our model going. In this case we don't want to run the models for too long so we will only use 2 chains. As way to keep a general eye on all our models MulTree will check whether the effective sample size (ESS) of each model is above some number across all parameters.

```
mulTree(mulTree.data = mulTree_data, formula = mul_formula, priors = mul_priors,
parameters = mul_parameters, output = "longevity_example", ESS = 1000,
chains = 2)
```

```
##
## 2016-07-14 - 17:57:39: MCMCglmm performed on tree 1
## Convergence diagnosis:
## Effective sample size is > 1000: TRUE
## 1956.574; 1754.576; 2000; 2000; 2000; 2000
## All levels converged < 1.1: TRUE
## 1.000927; 1.000777; 1.000416
## Individual models saved as: longevity_example-tree1_chain*.rda
## Convergence diagnosis saved as: longevity_example-tree1_conv.rda
##
## 2016-07-14 - 17:59:04: MCMCglmm performed on tree 2
## Convergence diagnosis:
## Effective sample size is > 1000: TRUE
## 2000; 2000; 1865.545; 2000; 2000; 1814.351
## All levels converged < 1.1: TRUE
## 1.000137; 1.001044; 0.9996352
## Individual models saved as: longevity_example-tree2_chain*.rda
## Convergence diagnosis saved as: longevity_example-tree2_conv.rda
##
## 2016-07-14 - 17:59:04: MCMCglmm successfully performed on 2 trees.
## Total execution time: 2.925504 mins.
## Use read.mulTree() to read the data as 'mulTree' data.
## Use summary.mulTree() and plot.mulTree() for plotting or summarizing the 'mulTree' data.
```

Now that we have run the model lets read the trees back in.

```
## Reading only one specific model
one_model <- read.mulTree("longevity_example-tree1_chain1", model = TRUE)
## This model is a normal MCMCglmm object that has been ran on one single tree
class(one_model) ; names(one_model)
```

```
## [1] "MCMCglmm"
```

```
##  [1] "Sol"        "Lambda"     "VCV"        "CP"         "Liab"
##  [6] "Fixed"      "Random"     "Residual"   "Deviance"   "DIC"
## [11] "X"          "Z"          "ZR"         "XL"         "ginverse"
## [16] "error.term" "family"     "Tune"
```

```
## Reading the convergence diagnosis test to see if the two chains converged for
## each tree
read.mulTree("longevity_example", convergence = TRUE)
```

```
## $`longevity_example-tree1_conv`
## Potential scale reduction factors:
##
##               Point est. Upper C.I.
## (Intercept)            1       1.00
## mass                   1       1.01
## volantvolant           1       1.00
##
## Multivariate psrf
##
## 1
##
## $`longevity_example-tree2_conv`
## Potential scale reduction factors:
##
##               Point est. Upper C.I.
## (Intercept)            1          1
## mass                   1          1
## volantvolant           1          1
##
## Multivariate psrf
##
## 1
```

```
## As indicated here, the chains converged for both chains!
## Reading all the models to perform the MCMCglmm analysis on multiple trees
all_models <- read.mulTree("longevity_example")
str(all_models)
```

```
## List of 5
##  $ (Intercept)          : num [1:8000] 0.315 -0.135 -1.171 -0.243 0.424 ...
##  $ mass                 : num [1:8000] 0.604 0.468 0.493 0.556 0.586 ...
##  $ volantvolant         : num [1:8000] 1.126 1.096 1.125 0.882 0.955 ...
##  $ phylogenetic.variance: num [1:8000] 1.102 1.051 1.007 0.712 1.136 ...
##  $ residual.variance    : num [1:8000] 0.0334 0.0248 0.0341 0.0605 0.0421 ...
##  - attr(*, "class")= chr "mulTree"
## This object contains 39600 estimations of the Intercept and the terms!

## If you want to remove the chains from the current directory run
#file.remove(list.files(pattern="longevity_example"))
## However when doing your actual analysis you sould keep all your models stored somewhere!
```

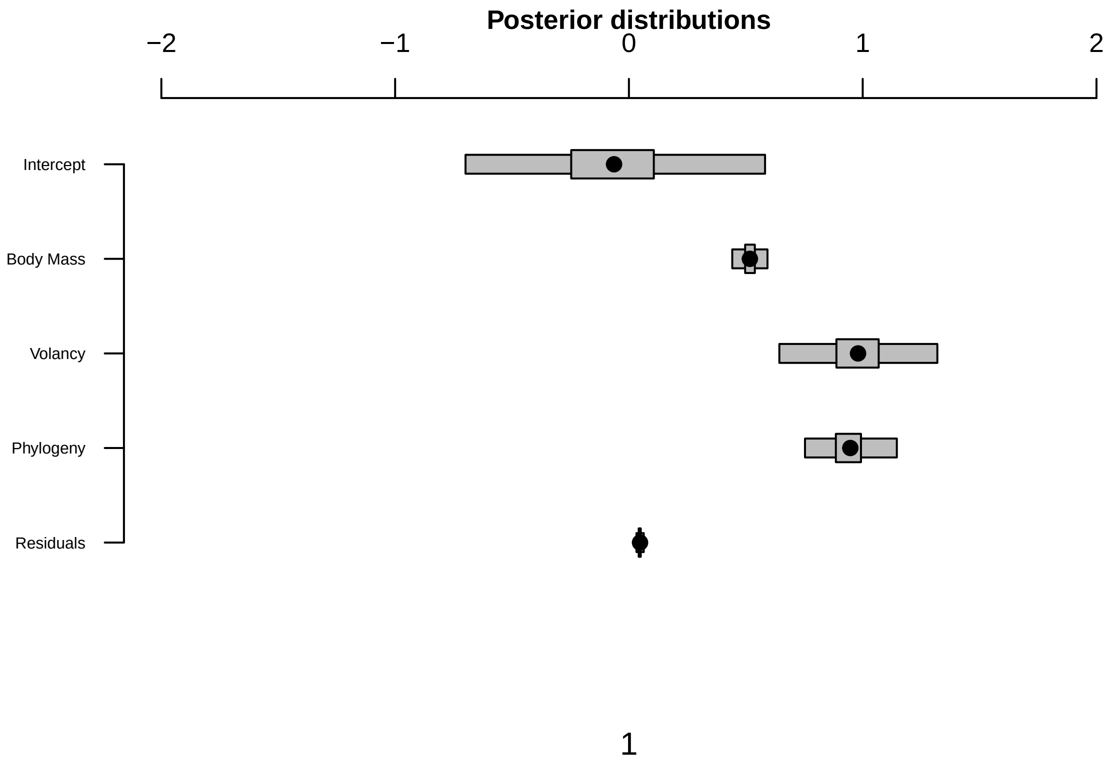
Great it looks similar to what we've seen before.

```
summarised_results <- summary(all_models, use.hdr = FALSE, cent.tend = mean, prob = c(75, 25))
```

And just for fun we can make some density plots

```
#quartz(width = 10, height = 5) ; par(mfrow = (c(1,2)), bty = "n")

plot(summarised_results, horizontal = TRUE, ylab = "", cex.coeff = 0.8,
main = "Posterior distributions", ylim = c(-2,2), cex.terms = 0.5,
terms = c("Intercept", "Body Mass", "Volancy", "Phylogeny", "Residuals"),
col = "grey", cex.main = 0.8)
```



Extending MCMCglmm: Imputing data

Another neat abilty of using MCMCglmm is the abilty to impute data. Imputation is essentially estimating some values for missing data using information from other linked data. This could be as simple as using the mean of some distribution or more usefully using the some corelative structure such as calulating expected values in a glm. However MCMCglmm will automatically impute missing data in the response variable allowing you carry out an analysis with some missing data.

```
#quartz(width = 10, height = 5) ; par(mfrow = (c(1,2)), bty = "n")

#missing data for ten and a 100 random species
index10 <- round(runif(10,1,length(mcmc_data$longevity)))
index100 <- round(runif(100,1,length(mcmc_data$longevity)))

#And replace them with NA
lifespan10 <-  mcmc_data$longevity
lifespan100 <-  mcmc_data$longevity
lifespan10[index10] <- NA
lifespan100[index100] <- NA

####and create a new dataframe to keep things clean
mcmc_data <- data.frame(lifespan10,lifespan100,mcmc_data)
```

Like before we need a prior

```
prior3 <- list(R = list(V=1, nu=0.002),
              G = list(G1 = list(V=1, nu=0.002)))
```

and some paramters

```
#no. of interations
nitt3 <- c(12000)
#length of burnin
burnin3 <- c(2000)
#amount of thinning
thin3 <- c(5)
```

Now we can run our models with missing models. WE dont need to but if we want to record the imputed lifespans of the missing species we can use the pl = TRUE argument.

```
mcmc_imput_10 <- MCMCglmm(fixed =  lifespan10 ~ mass + volant,
                     random= ~ animal,
                     family="gaussian",
                     pedigree = mcmc_tree,
                     data = mcmc_data,
                     nitt = nitt3,
                     burnin = burnin3,
                     thin = thin3,
                     prior = prior3,
                     verbose=FALSE,
                     pl = TRUE
                     )
summary(mcmc_imput_10)
```

```
##
##  Iterations = 2001:11996
##  Thinning interval  = 5
##  Sample size  = 2000
##
##  DIC: 23.68114
```

```
##
##  G-structure:  ~animal
##
##        post.mean l-95% CI u-95% CI eff.samp
## animal    0.6077   0.4106   0.8415     1030
##
##  R-structure:  ~units
##
##       post.mean l-95% CI u-95% CI eff.samp
## units   0.03964  0.02006  0.06217    984.2
##
##  Location effects: lifespan10 ~ mass + volant
##
##              post.mean l-95% CI u-95% CI eff.samp  pMCMC
## (Intercept)    0.07875 -0.71459  0.87343     2000  0.838
## mass           0.40697  0.24646  0.54985     1836 <5e-04 ***
## volantvolant   0.98152  0.07454  1.91093     2000  0.041 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

and now with 100 missing species

```
mcmc_imput_100 <- MCMCglmm(fixed =  lifespan100 ~ mass + volant,
                     random= ~ animal,
                     family="gaussian",
                     pedigree = mcmc_tree,
                     data = mcmc_data,
                     nitt = nitt3,
                     burnin = burnin3,
                     thin = thin3,
                     prior = prior3,
                     verbose=FALSE,
                     pl = TRUE
                     )
summary(mcmc_imput_100)
```

```
##
##  Iterations = 2001:11996
##  Thinning interval  = 5
##  Sample size  = 2000
##
##  DIC: 2.801723
##
##  G-structure:  ~animal
##
##        post.mean l-95% CI u-95% CI eff.samp
## animal    0.5827   0.3247   0.9176    304.1
##
##  R-structure:  ~units
##
##       post.mean l-95% CI u-95% CI eff.samp
## units   0.03824 0.009921   0.0799    422.5
##
```

```
##  Location effects: lifespan100 ~ mass + volant
##
##               post.mean l-95% CI u-95% CI eff.samp  pMCMC
## (Intercept)     0.1159  -0.6722   0.9325   1559.2  0.787
## mass            0.4161   0.2267   0.5956    638.0 <5e-04 ***
## volantvolant    0.4939  -0.5384   1.5429    688.6  0.357
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Even with 100 missing values in the response variable our model runs pretty well. Most of this is lilkey to be down to the data missing in a random fashion. If say however all the flying species were missing we simply could not run the model.

Since we recorded the impututed estimates we can also get values of lifespans for these species which are stored under the heading Liab

```
##as the Liab section stores what the values are for each species along each iteraction
##(even if the values are fixed) we will only look at the first 10
#notice that
head(mcmc_imput_100$Liab)[,1:10]
```

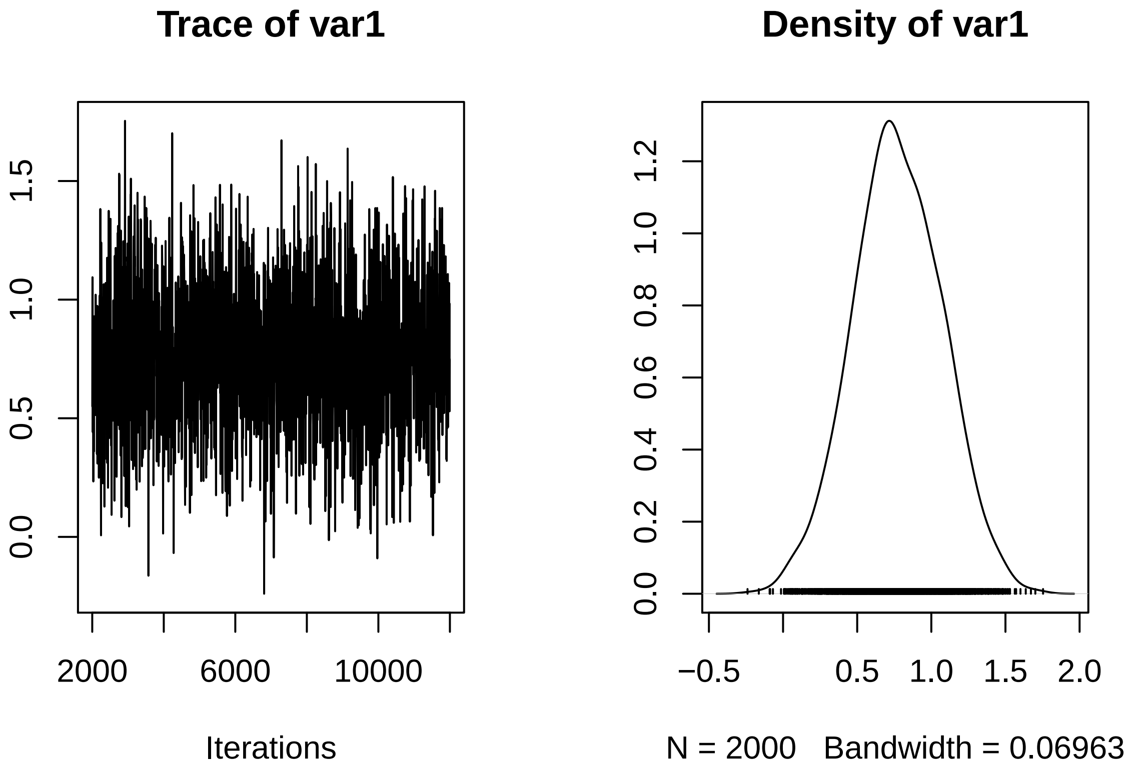
```
## Markov Chain Monte Carlo (MCMC) output:
## Start = 2001
## End = 2031
## Thinning interval = 5
##           [,1]       [,2]       [,3]       [,4]      [,5]      [,6]
## [1,] -2.056791 -1.6540739 -2.1256746 -1.8790019 -2.133185 -0.190961
## [2,] -2.056791 -1.6102240 -1.8200446 -1.5644328 -2.133185 -0.190961
## [3,] -2.056791 -2.1559792 -1.6191859 -1.6402943 -2.133185 -0.190961
## [4,] -2.056791 -0.7967968 -1.5214966 -2.0294613 -2.133185 -0.190961
## [5,] -2.056791 -1.3521434 -1.2660879 -1.5300953 -2.133185 -0.190961
## [6,] -2.056791 -1.2618198 -0.7747733 -0.4734427 -2.133185 -0.190961
## [7,] -2.056791 -1.3946747 -1.3265351 -0.9873907 -2.133185 -0.190961
##            [,7]       [,8]      [,9]        [,10]
## [1,] -0.5760142 -0.4205487 0.9929849  0.213283843
## [2,] -0.5238775 -0.4205487 0.9929849  0.692024924
## [3,] -0.2727500 -0.4205487 0.9929849  0.996143714
## [4,] -0.5613746 -0.4205487 0.9929849  0.938401662
## [5,] -0.6822195 -0.4205487 0.9929849  0.364776133
## [6,] -0.4783550 -0.4205487 0.9929849  0.830563024
## [7,]  0.0658650 -0.4205487 0.9929849 -0.009888513
```

The enties along the column are in the same order as the species along our row so if we wnat the estimate for the first species.

```
####plot of the first missing species
rownames(mcmc_data[index10[1],])
```

```
## [1] "Bison_bison"
```

```
plot(mcmc_imput_10$Liab[,index10[1]])
```



```
####mean of the first missing species
mean(mcmc_imput_10$Liab[,index10[1]])
```

```
## [1] 0.7735602
```

```
###and check it with the actual value
mcmc_data[index10[1],c("longevity")]
```

```
## [1] 1.108498
```

Imputation can be used in any of the models shown before. One of the major advangates is that instead of imputing using models without phylogentic corrections MCMCglmm allows this along with any number of random terms.

As a shamless plub I used this implementtion to develop and package that estimates Trophic discrimination factors for the use in stable isotopes. If intrested heres a step through guide and the pre-print of the paper.