Instalacja R

Najnowszą wersję R można ściągnąć ze strony

`http://cran.r-project.org/bin/windows/base`

Po zainstalowaniu w Menu Start pojawi się grupa 'R' pozwalająca na uruchomienie konsoli R. Praca w samej konsoli przypomina pracę w DOSie – wpisujemy komendy i po zatwierdzeniu klawiszem enter są one wykonywane. Niestety – z podstawową dystrybucją R dostajemy jedynie najczęściej używane pakiety i procedury statystyczne. Przeprowadzimy za ich pomocą większość testów statystycznych i stworzymy wykresy, są tam nawet funkcje służące do dopasowywania prostych modeli liniowych (`glm`, `lm`, `lme4`).

Aby dowiedzieć się czegokolwiek o dowolnym pakiecie (lub w ogólności – obiekcie w R) możemy użyć funkcji

```
>help(stats)
```

lub po prostu

```
>?stats
```

Co zrobić gdy potrzebujemy konkretnego pakietu a nie ma go w podstawowej dystrybucji? Używamy wtedy funkcji

```
>install.packages('MCMCglmm')
```

Instrukcja to sprawi, że R połączy się ze swoim repozytorium (poprzez wybrany przez nas serwer) i zainstaluje najnowszą wersję pakietu MCMCglmm (z którym będziemy pracować dzisiaj).

W trakcie pracy z R możemy zechcieć ją skończyć. Wydajemy wtedy polecenie

```
>q()
```

co zamyka konsolę. Przed zakończeniem program pyta nas czy zapisać zmiany. Jeśli zgodzimy się, utworzone zostaną 2 pliki: `.Rdata`, zawierający wszystkie utworzone obiekty i zmienne, oraz `.Rhistory` zawierający wszystkie wydane komendy. Przywrócenie ich jest możliwe za pomocą poleceń `load(file=".Rdata")` oraz `loadhistory(file=".Rhistory")` – pod jednym jednakże warunkiem: R pracuje w tym samym katalogu roboczym co w poprzedniej, zakończonej sesji. O tym jaki katalog jest roboczym przekonać się można wydając polecenie

```
>getwd()
```

Do dobrych zwyczajów w trakcie pracy z R należy tworzenie osobnych katalogów roboczych dla poszczególnych analiz – wtedy po uruchomieniu R po prostu przechodzimy do tego katalogu jako do katalogu roboczego:

```
>setwd(„C:/R/szkolenie")
```

i pracujemy w jego wnętrzu, co pozwala m.in. uniknąć podawania za każdym razem pełnej ścieżki do plików. W tym katalogu zapiszą się też wspomniane pliki historii etc.

Formula

Zanim przejdziemy do bardziej zaawansowanych technik statystycznych – warto poznać sposób zapisu modeli liniowych w R.

```
>formula <- y ~ 1+factor1*factor2+factor3+factor3:factor1
```

Powyższe polecenie powoduje **przypisanie** do zmiennej `model` obiektu o typie *formula*. Operator '`<-`' jest równoznaczny z '`=`' w innych językach (i w R działa z nim zamiennie). Co do samej formuły:

`y` – zmienna zależna; może być zestawem 2 lub więcej zmiennych – o tym szczegółowo niżej

`factorX` – czynniki kategoryczne lub kowariaty

`factor1*factor2` – dopasuj obydwa czynniki wraz z ich interakcją

`factor3:factor1` – dopasuj tylko interakcję obydwu czynników

`1` – *intercept*; jeśli pominięta – interecept dopasowywany jest domyślnie; czasami dla wygodniejszej interpretacji dobrze jest usunąć intercept (czyli zapisać `-1`) –o tym także niżej.

Zaawansowane specyfikacje formuł omówiono przy okazji modeli dwuzmiennowych.

Typy danych w R

Podstawowym typem danych w R jest wektor – np.:

```
> c(1,2,3,4,5,6)

[1] 1 2 3 4 5 6
```

tworzy wektor złożony z 6 liczb. Elementami wektora mogą być też zmienne znakowe (litery/słowa) oraz zmienne logiczne (prawda/fałsz). Warto pamiętać, że nawet pojedyncza zmienna jest także jednoelementowym wektorem a nie skalarem.

Inne typy danych to:

`array` – struktura wielowymiarowa – tablica obiektów o n wymiarach;

`matrix` – dwuwymiarowa tablica obiektów;

`list` – pozwala na przechowywanie obiektów różnych typów (np. tekst i liczby),czego powyższe nie potrafią;

`data frame` – najbardziej nas interesująca struktura; tablica dwuwymiarowa, gdzie kolumny reprezentują zmienne zaś wiersze poszczególne punkty danych.

Wczytywanie danych

Do pobierania danych służy ogólna funkcja read.table(). Działa ona tak:

```
> data <- read.table(file="plik.txt", header=TRUE, sep="\t")
```

Powyższe polecenie tworzy obiekt `data` o typie `data frame` i przypisuje mu dane odczytane z pliku `plik.txt`. Pierwsza linijka pliku jest użyta do nadania nazw zmiennym (argument `header`), zaś poszczególne pola danych (wartości zmiennych dla każdego punktu danych) oddzielone są od siebie tabulatorem (argument `sep`). Do poszczególnych kolumn, wierszy i danych obiektu data można mieć łatwy dostęp. Poniższe komendy np. umożliwiają dostęp do, kolejno: (1) pierwszej kolumny danych (cała pierwsza zmienna), (2) czwartego wiersza (wartości wszystkich zmiennych dla czwartego rekordu), (3) wszystkich rekordów danych w których wartość piątej zmiennej jest równa 44.

```
> data[,1]

> data[4,]

> data[,data[5]==44]
```

Do zmiennych można też odwoływać się przez ich nazwy, np:

```
> data$Chromosome.Number
```

Polecam lekturę funkcji `read.table`, `write.table` oraz funkcji odpowiadających podstawowym typom danych (np. `matrix()` lub `list()`).

Modele mieszane w MCMCglmm

Animal model

Zaczniemy od załadowanie pakietu MCMCglmm oraz wczytania przykładowych danych które dostarczone są z tym pakietem.

```
> library("MCMCglmm")
> data(BTdata)
> data(BTped)
```

Dane BTdata zawierają skok i kolor grzbietu sikory modrej, zmierzone dla ponad 800 osobników (*animal*),wraz z danymi matki (*dam*), gniazdem wychowywania, datą klucia oraz płcią. Zmienne liczbowe są przedstawione jako odchylenia od średniej. Dane BTped z kolei zawierają dane wszystkich osobników, wraz z ich matkami (*dam*) oraz ojcami (*sire*). Istotne jest, że wszystkie osobniki będące ojcami/matkami są też w kolumnie animal. W drugą stronę nie musi to działać – jeśli rodzice osobnika nie są znani – wpisuje się po prostu NA. Ponieważ wśród osobników są takie, które nie zostały oznaczone pod względem płci – możemy oczyścić dane aby takich osobników nie było:

```
> BTdata<-BTdata[!BTdata$sex=="UNK",]
```

Stwórzmy prosty model liniowy (GLM) z dwoma czynnikami, ustalonym płci oraz kowariatą daty klucia; uwzględniamy także interakcję tych dwóch czynników. Zmienną zależną będzie długość skoku. Aby uruchomić taki model piszemy:

```
> model1 <- MCMCglmm(tarsus~1+sex*hatchdate, data=BTdata,
+ verbose=FALSE)
```

Po uruchomieniu programu i odczekaniu chwili R powraca do stanu 'zachęty'. Oznacza to, że obiekt `model1` zawiera rozwiązanie zadanego modelu i możemy przystąpić do jego analizy. Aby dostać się do rozwiązań efektów ustalonych piszemy:

```
> posterior.mode(model1$Sol)
```

To, co dostajemy w wyniku zastosowania łańcucha Markova to rozkłady *a posteriori* wszystkich czynników zawartych modelu. Przekonać się o tym można wyświetlając je na ekranie:

```
> plot(model1$Sol)
```

Jak łatwo zauważyć – nie widzimy tutaj nigdzie wartości P pozwalającej powiedzieć cokolwiek o istotności danego efektu. Ponieważ jednak wartość estymowane przez nasz model są w istocie odchyleniami w danej grupie od średniej ogólnej (intercept) – jeśli nie są istotnie różne od zera – wskazują na brak istotności danego czynnika modelu. W naszym przypadku 'na oko' widać, że efekt daty klucia oraz interakcja daty klucia z płcią nie mają istotnego wpływu na długość skoku sikory modrej. Upewnić się o tym możemy wyświetlając przedziały ufności dla otrzymanych estymatorów:

```
> HPDinterval(model1$Sol)
```

Przedziały ufności dla daty klucia oraz interakcji obejmują zero – co potwierdza wcześniejsze spostrzeżenia.

Co z efektami losowymi? Tutaj sprawa jest trudniejsza i wymaga wprowadzenia pojęcia rozkładu *a priori*. Jest to jeden elementów statystyki bayesowskiej i nie wdając się w szczegóły sprowadza się do obliczania rozkładu *a posteriori* estymowanej wartości na podstawie nie tylko danych pomiarowych ale naszego wstępnego 'poglądu' na to jaką wartość estymowany parametr powinien mieć. Innymi słowy – na podstawie wcześniejszych doświadczeń – tworzymy rozkład *a priori* estymatora i razem z danymi pomiarowymi bierze on udział w wyliczeniu rozkładu *a posteriori*. Sam rozkład a priori – poza wartością, w którą 'wierzymy', uwzględnia też nasze zaufanie do takiej a nie innej wartości parametru (może ten rozkład być wręcz płaski – tzw. *uniform dustribution*) kiedy każda wartość estymatora jest równie prawdopodobna (najsłabsze 'przekonanie' co do prawidłowości którejkolwiek z wartości).

Dla efektów ustalonych rutynowo nie definiuje się wartości a priori (domyślne mają średnią zero i bardzo dużą wariancję, z niewielkim parametrem wiarygodności, co sprawia że o wartości

estymatora decydują niemal całkowicie dane). W przypadku efektów losowych wybór wartości a priori jest znacznie bardziej istotny. W przypadku, gdy dane są dobre i jest ich dużo – wartości tzw. priorów nie wpłyną na oszacowanie komponent wariancji. Z tego powodu najczęściej warto wartości a priori komponent wariancji ustawić jako równe (w stosunku do ilości czynników losowych + 1 dla czynnika błędu) proporcje ogólnej wariancji w danej zmiennej zależnej. W pakiecie MCMCglmm priory definiuje się jako listy list zawierające wartości a priori wariancji dla każdego z czynników losowych oraz parametr 'wiarygodności' mówiący na ile ufamy wartości priora. Parametr ten ma z reguły wartość n, gdzie n jest liczbą zmiennych zależnych w modelu. Stosując takie n dostajemy tzw. właściwy prior o najniższym możliwym poziomie wiarygodności – czyli to, o co nam chodzi ale o poprawnym rozkładzie prawdopodobieństwa (w skrócie – całkującym się do jedności).

Dodajmy do modelu 1 czynnik losowy – gniazdo wychowywania. Oto prior dla takiego modelu:

```
> prior <- list(R=list(V=var(BTdata$tarsus, na.rm=TRUE)/2,n=1),
+ G=list(G1=list(V=var(BTdata$tarsus, na.rm=TRUE)/2,n=1)))
```

Wnioskowanie dotyczące efektów ustalonych jest podobne – co z efektami losowymi? W ich przypadku stosowanie przedziałów ufności daje jedynie przybliżoną odpowiedź. Wynika to z faktu, że wartość komponent wariancji jest wymuszana jako dodatnia w MCMCglmm. Innymi słowy – przewidywana wariancja może swoim przedziałem ufności nie obejmować zera z tego właśnie powodu. Właściwym sposobem testowania istotności czynników losowych jest stosowanie DIC (Deviance Information Criterion). Rozważmy 2 modele: 1 z czynnikiem fosternest, drugi bez niego. Każdy z tych modeli posiada parametr DIC mówiący o jakości jego dopasowania do danych. Porównajmy je:

```
> model1$DIC #model bez danego czynnika losowego
[1] 2102.911
> model2$DIC #model z czynnikiem losowym
[1] 2019.207
```

DIC modelu drugiego jest niższe – wskazuje to na lepsze dopasowanie takiego modelu w stosunku do modelu bez danego czynnika. Oznacza to, że wariancja związana z gniazdem wychowywania jest istotnie różna od zera.

W jaki sposób z pokazanego modelu zrobić typowy animal model? Należy zdefiniować model tak, by macież wariancji/kowariancji resztowych nie była macierzą jednostkową tylko macierzą odzwierciedlającą pokrewieństwa między osobnikami. Osiągnąć to można dodając do listy elementów losowych dodatkowy czynnik *animal* skojarzony z odpowiednią kolumną danych, oraz definiując plik genealogii (argument *ped*) pozwalający określić pokrewieństwa między osobnikami. Czynnik animal będzie zawierał addytywną wariancję genetyczną badanej cechy. Jego uwzględnienie wymaga oczywiście zdefiniowania nowego priora – uwzględniającego 3 czynniki losowe:

```
prior2 <- list(R=list(V=var(BTdata$tarsus, na.rm=TRUE)/3,n=1),
+ G=list(G1=list(V=var(BTdata$tarsus, na.rm=TRUE)/3,n=1),
+ G2=list(V=var(BTdata$tarsus,na.rm=TRUE)/3,n=1)))
```

```
> animalmodel <- MCMCglmm(tarsus~sex*hatchdate,
+ random=~animal+fosternest, ped=BTped, data=BTdata, verbose=FALSE,
+ prior=prior2)
```

Spojrzenie na rozkłady poszczególnych komponent wariancji wskazuje na dużą istotność zarówno gniazda wychowywania jak i czynnika genetycznego. Użyteczną własnością rozkładów *a posteriori* jest łatwość ich przekształcania. Możemy np. bardzo łatwo obliczyć odziedziczalność danej cechy i jej przedział ufności.

```
> h2 <- animalmodel$VCV[,1]/(animalmodel$VCV[,1]+
+ animalmodel$VCV[,2]+animalmodel$VCV[,3])
> posterior.mode(h2) #najbardziej prawdopodobna wartość h2
    var1
[1] 0.53214
> HPDinterval(h2) #przedział ufności dla h2
         lower     upper
var1 0.3241312 0.6893012
attr(,"Probability")
[1] 0.95
```

W identyczny sposób przeprowadza się analizy filogenetyczne. Jedyną różnicą jest to, że miejsce osobników zajmują taksony a miejsce genealogii zajmuje plik opisujący pokrewieństwa filogenetyczne między tymi taksonami (o czym niżej).

Diagnoza modeli

Metody oparte na łańcuchach Markova są metodami stochastycznymi. Tak jak z losowaniem kul z urny – jeśli nie zamieszamy wystarczająco dobrze, kolejne losowania nie będą niezależne od siebie. W analizie opartej na MCMCglmm taką sytuację określamy jako autokorelację. Wykrycie jej możliwe jest już po pierwszym spojrzeniu na wykresy serii czasowych: jeśli pojawia się w nich stały trend (linia biegnąca przez środek serii nie jest idealnie pozioma) – autokorelacja istnieje i takim wynikom nie powinniśmy do końca ufać. O istnieniu autokorelacji przekonać się także można stosując wbudowaną funkcję:

```
> autocorr(animalmodel$VCV)
, , animal

            animal   fosternest        units
Lag 0    1.0000000 -0.193219369 -0.851977627
Lag 10   0.7458187 -0.178167941 -0.715020058
Lag 50   0.3678501 -0.108362688 -0.348028910
Lag 100  0.1877636 -0.040664729 -0.190185747
Lag 500 -0.0133513  0.002681355  0.001437267

, , fosternest

              animal  fosternest        units
Lag 0   -0.193219369  1.00000000  0.108742947
```

```
Lag 10  -0.163259258  0.18090504  0.128767401
Lag 50  -0.087502979 -0.03196609  0.078660617
Lag 100  0.001148218 -0.02663029  0.003201562
Lag 500  0.010453338 -0.01498867 -0.007214128

, , units

               animal  fosternest       units
Lag 0   -0.8519776271  0.10874295  1.00000000
Lag 10  -0.7168096558  0.15160465  0.69954005
Lag 50  -0.3429159550  0.11116653  0.31187333
Lag 100 -0.1797198995  0.02826373  0.20162321
Lag 500  0.0002941358 -0.01362942 -0.01851532
```

Porównanie wartości autokorelacji dla czynnika animal oraz units (ten czynnik odpowiada wierszom w tabeli danych, jest to więc czynnik błędu – *residual*) wskazuje, że ta pierwsza jest o 2 rzędu wielkości większa, co w normalnej analizie nie powinno być akceptowane. Z autokorelacjami najłatwiej sobie poradzić uruchamiając łańcuch na dłużej, wydłużając tą część łańcucha z której nie są pobierane próby do rozkładu końcowego (tzw. Wstępna faza mieszania) oraz pobierając próby rzadziej. Domyślnie łańcuchy składają się z 13000 iteracji, faza wstępna obejmuje 2000 iteracji, próby pobierane są co 30 iteracji. Rozsądnie jest wydłużyć ten czas do 100000 iteracji, z fazą wstępną 30000 i odstępem między próbami około 70-100:

```
> animalmodel <- MCMCglmm(tarsus~sex*hatchdate,
+ random=~animal+fosternest, ped=BTped, data=BTdata, verbose=FALSE,
+ prior=prior2, nitt=100000, burnin=30000, thin=80)
```

Drugim ważnym elementem diagnostyki modelu jest upewnienie się, że zastosowany prior nie ma wpływu na analizę. Najłatwiej zrobić to uruchamiając alternatywny model ze zmienionym priorem; możemy np. zamiast dzielić wariancję po równo, stworzyć prior z nieproporcjonalnie większym udziałem kontroli genetycznej:

```
prior2alt <- list(R=list(V=var(BTdata$tarsus, na.rm=TRUE)*0.1,n=1),
+ G=list(G1=list(V=var(BTdata$tarsus, na.rm=TRUE)*0.8,n=1),
+ G2=list(V=var(BTdata$tarsus,na.rm=TRUE)*0.1,n=1)))
```

Warto także pamiętać, że MCMCglmm pozwala także łatwo dopasowywać modele z powtarzanymi pomiarami. Wystarczy jedynie jako czynnik losowy uwzględnić jednostkę (np. organizm) poddawaną kilkukrotnym pomiarom (np. w ciągu jakiegoś czasu). Po szczegóły odsyłam do 3 części Tutoriala (Wilson et al. 2010. J.Anim. Ecol.)

Model dwuzmiennowy (*bivariate*)

Aby móc oszacować prawidłowo korelacje genetyczne, lub np. wynikające z istnienia efektów matczynych między cechami należy zdefiniować modele w których obydwie cechy fenotypowe znajdują się po stronie zmiennych zależnych. W modelach takich przewidywanymi wartościami nie są wariancje odpowiadające czynnikom losowym ale całe macierze zawierające odpowiednie

wariancje i kowariancje, mające wymiar taki jak liczba badanych cech. Sprawdźmy, czy istnieje korelacja genetyczna między długością skoku a kolorem płaszcza u sikor. Prior potrzebny do tego modelu musi zawierać nie pojedyncze wariancje a macierze (ko)wariancji między tymi dwiema cechami:

```
> phen.var <- matrix(c(var(BTdata$tarsus,
na.rm=TRUE),0,0,var(BTdata$back, na.rm=T)),2,2)
> phen.var
         [,1]     [,2]
[1,] 1.008650 0.000000
[2,] 0.000000 0.995066
```

Zauważmy, że kowariancje zostały pozostawione jako zera (brak korelacji). A oto model:

```
> priorbi <- list(R=list(V=phen.var/3,n=2),
+ G=list(G1=list(V=phen.var/3,n=2),
+ G2=list(V=phen.var/3,n=2)))
> animalmodelbi <- MCMCglmm(cbind(tarsus,back)~trait-1+sex,
+                   random=~us(trait):animal+idh(trait):fosternest,
+                   rcov=~us(trait):units, pedigree=BTped,
+                   family=c('gaussian','gaussian'),
+                   data=BTdata, prior=priorbi, verbose=F)

> plot(animalmodelbi$VCV)
> posterior.mode(animalmodelbi$VCV)
> HPDinterval(animalmodelbi$VCV)
```

Co oznaczają dodatkowe opcje w modelu? Argument `rcov` definiuje odpowiednią strukturę (ko)wariancji, `family` zaś definiuje rodzaj rozkładu jaki posiada każda ze zmiennych zależnych (mogą to być różne rozkłady). Co natomiast oznaczają skróty `us` oraz `idh`? Definiują one rodzaj struktury (ko)wariancji jaki zastosowany jest do danego czynnika losowego. Struktura typu `us` pozwala na oszacowanie zarówno wariancji jak kowariancji, natomiast struktura typu `idh` blokuje kowariancje na wartości zerowej i nie oszacowuje ich (stosujemy ją gdy np. układ eksperymentalny wyklucza korelację między dwoma cechami na jakimś poziomie). Zauważmy także, że model wykorzystuje trzeci wbudowany typ zmiennej (obok `units` oraz `animal`) – zmienną `trait`. Zmienna ta odpowiada kolumnom zmiennych zależnych. Użycie jej jako efektu losowego pozwala oszacować wpływ płci na każdą ze zmiennych zależnych osobno, zaś wyrzucenie głównego intercept (`-1`) pozwala obliczyć dwie osobne średnie zamiast jednej średniej dla zmiennej tarsus, i kontrastu dla zmiennej back. Wykorzystanie `trait` w efektach losowych pozwala z kolei na stworzenie odpowiedniej struktury wariancji –zauważmy, że jest to formalnie interakcja (dwukropek!) z narzuconym typem macierzy (ko)wariancji.

Podobnie jak w przypadku odziedziczlaności – możemy łatwo obliczyć korelację gnetyczną (związaną z czynnikiem animal) I jej przedział ufności.

```
> rG <-
animalmodelbi$VCV[,2]/sqrt(animalmodelbi$VCV[,1]*animalmodelbi$VCV[,
4])
```

```
> posterior.mode(rG)
      var1
-0.3919104
> HPDinterval(rG)
          lower      upper
var1 -0.6263184 0.04677056
attr(,"Probability")
[1] 0.95
```

W analogiczny sposób możemy badać korelacje między większą liczbą cech – macierze wtedy rosną w postępie kwadratowym, musimy się także liczyć ze znacznym wzrostem długości obliczeń. Zaproponowany sposób budowania modeli nadaje się także (bardziej niż dotąd powszechnie stosowane metody oparte na interakcjach) do badania interakcji ze środowiskiem (GEI) czy płcią. W takich sytuacjach jako 2 osobne cechy traktuje się daną fenotypową cechę zmierzoną w 2 środowiskach lub 2 płciach. Należy jednak wtedy pamiętać, że struktura (ko)wariancji resztkowej jest inna: każdy osobnik jest mierzony tylko pod względem jednej 'cechy' (bo np. jest tylko samcem lub tylko samicą) – stąd nie możemy oczekiwać korelacji wariancji resztkowych i musimy dla czynnika błędu używać struktury wariancji typu `idh`. Obok dwóch przedstawionych – możliwe są inne struktury wariancji, pozwalające testować bardziej wyrafinowane hipotezy (patrz Overview).

Inne rozkłady – przykład danych binarnych

W przypadku danych binarnych pojawia się specyficzny problem – określenie wariancji błędu jest niemożliwe, jest ona całkowicie determinowana przez średnią (podobnie jest np. w przypadku rozkładu Poissona). W takiej sytuacji konstruując priors nadaje się takim 'niemożliwym' wariancjom arbitralna wartość i blokuje się je przy tej wartości:

```
> data(PlodiaRB)
> prior <- list(R=list(V=1,n=0,fix=1),G=list(G1=list(V=1,n=0)))
> modelbin <- MCMCglmm(Pupated~1,random=~FSfamily,
+ family='categorical', data=PlodiaRB, prior=prior, verbose=F)
```

Filogeneza

Użyjmy wbudowanych danych dot. pokrewieństwa między rzędami ptaków. Poniższy kod używa tych danych do wygenerowania drzewa z podkreślonymi na czerwono wybranymi liniami:

```
> data(bird.families)
> some.families<c("Struthionidae","Gruidae",
+ "Passeridae","Fringillidae")
> plot(bird.families,  cex=0.3, tip.col=c("black","red")
+ [bird.families$tip.label %in% some.families+1])
```

Możemy obejrzeć te pokrewieństwa w postaci macierzy (tutaj wyświetlany jest jej podzbiór; cała macierz byłaby zbyt wielka).

```
> Aphylo <- vcv.phylo(bird.families)[some.families, some.families]
> Aphylo
              Struthionidae Gruidae Passeridae Fringillidae
Struthionidae            28     0.0        0.0          0.0
Gruidae                   0    28.0        6.4          6.4
Passeridae                0     6.4       28.0         18.0
Fringillidae              0     6.4       18.0         28.0
```

Sam obiekt `bird.families` jest obiektem klasy `phylo` i jako taki może bezpośrednio być przekazany do MCMCglmm poprzez argument `pedigree`.