How to fit an animal model

An ecologist guide

Julien Martin

18-03-2021

Table des Matières

Preface

This book is a collection of tutorial from the excellent paper by (**?**). Instead of just copy pasting the tutorial in a bookdown format, the tutorials have been updated to work with the newest version of the softwares and extended to present other softwares. **However, this is still a work in progress.**

Contributors

List of people who contributed to update and extend tutorials:

- Eric Postma
- Julien Martin

Chapitre 1

Introduction

The book is provides a series of tutorials (and accompanying data files) to fit animal model in `R` using different packages (`ASReml-R`, `gremlin`, `MCMCglmm` and `brms`) . You will need to carefully follow the instructions below to download the data files and install the R packages. Before beginning the tutorial, we assume the reader has successfully installed the chosen R package on their computer and has saved the required data files to an appropriate directory from which they will be read. Full instructions for how to do this are provided with software distributions.

To work though the different tutorial I would recommend to create a folder where you will save your different `R` scripts for the tutorials. In addition, I recommand to create a subfolder `data` to save the files needed.

1.1 Data

1.1.1 Data files

You will need to download 3 data files for the tutorial in `R`:

- gryphon.csv: data on gryphon birth weight
- gryphonRM.csv: data
- gryphonped.csv

In addition, some models presented in the tutorials can take a while to run (sometimes $>$ 1 hour), thus we are also providing model the model outputs to allow you continue the tutorial without waiting for the model to run.

The files are available here

I recommend to save the data and Rdata files in a subfolder `data` in the folder you will use as your working directory for R and where you will save your R scripts. It should be noted that the tutorial are using this structure to read or save data.

1.1.2 Notes on data and pedigree

It is always important to take time to think carefully about the strengths and potential limitations of your pedigree information before embarking on quantitative genetic analyses. Pedigree Viewer, written by Brian Kinghorn, is an extremely useful application for visualising pedigrees, and can be downloaded from: `http://www-personal.une.edu.au/~bkinghor/pedigree.htm`. `Pedantics` an R package written by Michael Morrissey and distributed through CRAN (`http://cran.r-project.org/`) can also be used for this and offers some nice additional features for visualising pedigree structures and generating associated statistics. Before you begin running through the tutorials, we advise taking a moment to look at the pedigree files provided with them using Pedigree Viewer or Pedantics.

1.2 R

You should check that you have the most current version of R and R packages. You can check the number of the current version on CRAN. If you need to update (or install) R packages, use `install.packages()` and follow the prompted instructions.

1.2.1 R packages

1.2.1.1 asreml-r

ASReml-R is commercial software published by VSN international (`http://www.vsni.co.uk/software/asreml/`). This package is not free and requires a key access (add the price of it!)

1.2.1.2 gremlin

`gremlin` is a little monster appearing if you wet a mugwai and feed it after midnight. It is also a great and promising software to fit mixed models using a frequentist approach.

1.2.1.3 MCMCglmm

`MCMCglmm` is an R package for Bayesian mixed model analysis written by Jarrod Hadfield. It is freeware distributed through CRAN (`http://cran.r-project.org/`). Information about the package, together with a user manual and vignettes are available at `http://cran.r-project.org/web/packages/MCMCglmm/index.html` . Reference: (**?**, **?**).

This module provides some information that applies to MCMCglmm-based analyses in general, but that will not be included in other tutorials. Most importantly, this applies to some of the simplest waysof determining the performance of a run using MCMCglmm, i.e., verification of the validity of of the posterior distribution. This tutorial is not a substitute for working through the MCMCglmm course notes, which is available from CRAN (the Comprehensive R ArchiveNetwork, `http://cran.r-project.org/`, or can be accessed in R using the command vignette("CourseNotes","MCMCglmm")). These tutorials do not introduce one of the main advantages of using MCMCglmm for analyses of data from natural populations -the ability to properly model non-normal responses. These capabilities are introduced inthe documentation that is distributed with MCMCglmm, and available from CRAN.

1.2.1.4 brms

`brms` provides an interface to fit Bayesian generalized multivariate (non-)linear multilevel models using `Stan`, which is a C++ package for obtaining full Bayesian inference (see `https://mc-`

stan.org/). The formula syntax is an extended version of the syntax applied in the 'lme4' package to provide a familiar and simple interface for performing regression analyses.

It should be noted that if `brms` is able to fit animal model the parametrization used based on Kroeneker products is not the most efficient and can take quite longer than using a DAG parametrisation directly in `stan`.

Chapitre 2

Univariate animal model

This tutorial will demonstrate how to run a univariate animal model to estimate genetic variance in birth weight in the mighty gryphons.

2.1 Scenario and data

2.1.1 Scenario

In a population of gryphons there is strong positive selection on birth weight with heavier born individuals having, on average higher fitness. To find out whether increased birth weight will evolve in response to the selection, and if so how quickly, we want to estimate the heritability of birth weight.

2.1.2 Data files

Open `gryphonped.csv` and `gryphon.csv` in your text editor. The structure and contents of these files is fairly self-explanatory. The pedigree file `gryphonped.csv` contains three columns containing unique IDs that correspond to each animal, its father, and its mother. Note that this is a multigenerational pedigree, with the earliest generation (for which parentage information is necessarily missing) at the beginning of the file. For later-born individuals maternal identities are all known but paternity information is incomplete (a common situation in real world applications). The phenotype data, as well as additional factors and covariates that we may wish to include in our model are contained in `gryphon.csv`. Columns correspond to individual identity (`animal`), maternal identity (`mother`), year of birth (`byear`), sex (`sex`, where `1` is female and `2` is male), birth weight (`bwt`), and tarsus length (`tarsus`). Each row of the data file contains a record for a different offspring individual. Note that all individuals included in the data file must be included as offspring in the pedigree file.

We can read teh data file, using `read.csv()` which consider by default that `NA` is the symbol for missing values and that the first line of the file contains the column headers.

It is a good idea to make sure that all variables are correctly assigned as numeric or factors:

```
gryphon$animal <- as.factor(gryphon$animal)
gryphon$mother <- as.factor(gryphon$mother)
gryphon$byear <- as.factor(gryphon$byear)
gryphon$sex <- as.factor(gryphon$sex)
gryphon$bwt <- as.numeric(gryphon$bwt)
gryphon$tarsus <- as.numeric(gryphon$tarsus)
```

Similarly we can read in the pedigree file, using `read.csv()` which consider by default that `NA` is the symbol for missing values and that the first line of the file contains the column headers.

```
## 'data.frame':    1309 obs. of  3 variables:
##  $ id    : int  1306 1304 1298 1293 1290 1288 1284 1283 1282 1278 ...
##  $ father: int  NA NA NA NA NA NA NA NA NA NA ...
##  $ mother: int  NA NA NA NA NA NA NA NA NA NA ...
```

```
gryphonped$id <- as.factor(gryphonped$id)
gryphonped$father <- as.factor(gryphonped$father)
gryphonped$mother <- as.factor(gryphonped$mother)
```

Now that we have imported the data and the pedigree file, we are ready to fit an animal model.

2.2 Asreml-R

2.2.1 Running the model

First we need to load the `asreml` library:

```
library(asreml)
```

To be able to fit an animal model Asreml-r needs (the inverse of) the relationship matrix:

```
ainv <- ainverse(gryphonped)
```

We are now ready to specify our first model:

```
model1 <- asreml(
  fixed = bwt ~ 1, random = ~ vm(animal, ainv),
  residual = ~ idv(units),
  data = gryphon,
  na.action = na.method(x = "omit", y = "omit")
)
```

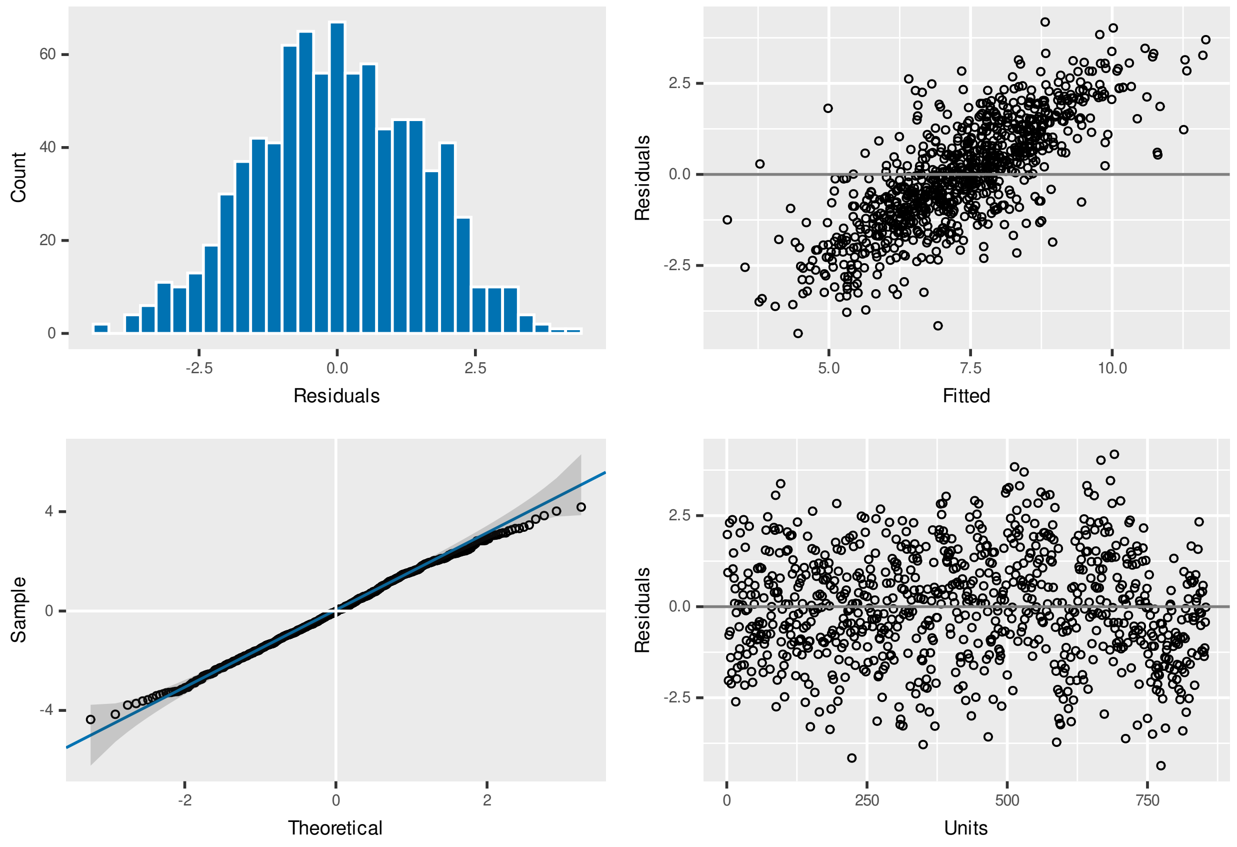
```
## Model fitted using the sigma parameterization.
## ASReml 4.1.0 Thu Mar 18 21:28:29 2021
##          LogLik        Sigma2     DF     wall    cpu
## 1      -4128.454          1.0    853 21:28:29    0.0
## 2      -3284.272          1.0    853 21:28:29    0.0
## 3      -2354.992          1.0    853 21:28:29    0.0
## 4      -1710.357          1.0    853 21:28:29    0.0
## 5      -1363.555          1.0    853 21:28:29    0.0
## 6      -1263.516          1.0    853 21:28:29    0.0
## 7      -1247.854          1.0    853 21:28:29    0.0
## 8      -1247.185          1.0    853 21:28:29    0.0
## 9      -1247.183          1.0    853 21:28:29    0.0
```

In this model, `bwt` is the response variable and the only fixed effect is the mean (the intercept, denoted as `1`). The only random effect we have fitted is `animal`, which will provide an estimate of V_A. Our random `animal` effect is connected to the inverse related matrix `ainv`. `data=` specifies the name of the dataframe that contains our variables. Finally, we tell `asreml()` what to when it encounters `NA`s in either the dependent or predictor variables (in this case we choose to remove the records). If you use the argument "include" instead of "omit", model will keep the NA. With x="include", the model will exchange `NA` with 0. Be careful you need to standardize your trait so the mean will be equal to 0, if not estimates (including covariance in multivaraite models) could be strongly biased due to the the missing values. y="include" will exchange `NA` with a factor labelled mv which will be included in the sparse equation. For more details see asreml-R manual.

A note of the specification of the structure of the residuals: This simple univariate model will run fine without `residual=~idv(units)`. However, if you are going to use `vpredict()` to calculate the heritability (see below), not specifying the residuals in this way will result in a standard error for the heritability that is incorrect.

Any model has assumption which need to be checked. The model can be plot which help visualizing the distribution of the model residual and check the different assumptions.

```
plot(model1)
```



To see the estimates for the variance components, we run:

```
summary(model1)$varcomp
```

```
##                     component std.error  z.ratio bound %ch
## vm(animal, ainv)  3.395398 0.6349915 5.347154     P   0
## units!units       3.828602 0.5185919 7.382687     P   0
## units!R           1.000000        NA       NA     F   0
```

We fitted a single random effect so have partitioned the phenotypic variance into two components. The `vm(animal, ainv)` variance component is V_A and is estimated as 3.4. Given that the ratio of V_A to its standard error (`z.ratio`) is considerably larger than 2 (*i.e.* the parameter estimate is more than 2 SEs from zero) this looks likely to be significant. The `units!units` component refers to the residual variance V_R, and `units$R` should be ignored. If you don't include `residual=~idv(units)`in your model specification, `units$R` will provide you with the residual variance.

2.2.2 Estimating heritability

We can calculate the h^2 of birth weight from the components above since $h^2 = V_A/V_P = V_A/(V_A+V_R)$. Thus according to this model, h^2 = 3.4 / (3.4 + 3.83) = 0.47.

Alternatively we can use the `vpredict()` function to calculate h^2 and its standard error:

```
vpredict(model1, h2.bwt ~ V1 / (V1 + V2))
```

```
##         Estimate         SE
## h2.bwt 0.4700163 0.07650881
```

2.2.3 Adding fixed effects

To add fixed effects to a univariate model simply modify the model statement. For example we might know (or suspect) that birth weight is a sexually dimorphic trait and therefore fit a model

```
model2 <- asreml(
  fixed = bwt ~ 1 + sex,
  random = ~ vm(animal, ainv),
  residual = ~ idv(units),
  data = gryphon,
  na.action = na.method(x = "omit", y = "omit")
)
```

```
## Model fitted using the sigma parameterization.
## ASReml 4.1.0 Thu Mar 18 21:28:30 2021
##           LogLik        Sigma2     DF     wall    cpu
##  1     -3364.126           1.0    852 21:28:30    0.0
##  2     -2702.117           1.0    852 21:28:30    0.0
##  3     -1978.916           1.0    852 21:28:30    0.0
##  4     -1487.834           1.0    852 21:28:30    0.0
##  5     -1236.350           1.0    852 21:28:30    0.0
##  6     -1172.771           1.0    852 21:28:30    0.0
##  7     -1165.270           1.0    852 21:28:30    0.0
##  8     -1165.093           1.0    852 21:28:30    0.0
##  9     -1165.093           1.0    852 21:28:30    0.0
```

Now we can look at the fixed effects parameters and assess their significance with a conditional Wald F-test:

```
summary(model2, coef = TRUE)$coef.fixed
```

```
##             solution std error  z.ratio
## sex_1       0.000000        NA       NA
## sex_2       2.206996 0.1619974 13.62365
## (Intercept) 6.058669 0.1718244 35.26082
```

```
wald.asreml(model2, ssType = "conditional", denDF = "numeric")
```

```
## Model fitted using the sigma parameterization.
## ASReml 4.1.0 Thu Mar 18 21:28:30 2021
```