Tutorial

Estimation of a biological trait heritability using the animal model and MCMCglmm (version 2)

Pierre de Villemereuil

June 26, 2023

Table of Contents

Preamble

Why this tutorial?

This tutorial is intended for students or researchers in evolutionary ecology or evolutionary quantitative genetics, interested in using the animal model to estimate the heritability of biological traits, with a focus in wild populations. It aims at providing theoretical and practical help on three main issues: (*i*) understanding what heritability is, what it quantifies and how the animal model works; (*ii*) learning by practice how to implement animal models using the MCMCglmm R package; and (*iii*) introducing Bayesian statistics (priors, Markov Chain Monte Carlo, etc.). The author tried to use examples of increasing complexity to start with some very easy "hands on" and finish with some of the most tedious aspects of MCMC estimation methods. Finally, although this tutorial is directly inspired from J. Hadfield course notes (Hadfield 2016), it tries to bring new information more focused on heritability estimation and good use of MCMC.

Why a new version?

The previous version was relatively old (written in 2012) and a part of the content was starting to be a bit outdated. Also, I changed my mind on a few advices of the first version (including fixed effects and dealing with non Gaussian traits) over time and wanted a new version to reflect the "state-of-the-art" I have in mind.

Prior knowledge

The readers are assumed to have an average knowledge of evolutionary biology and its problems, along with some notions of quantitative genetics. They are also assumed to have a basic understanding of statistical mixed models framework and its associated vocabulary (fixed and random effects especially) and to have some familiarity with their formulation. A vague knowledge of Bayesian statistics and a familiarity with the R statistical software are needed for a correct understanding of section 3, but not for the section 1 and section 2.

How to cite this document?

Since this tutorial is essential merging the old material of the first tutorial, some parts of the QGglmm vignette and the more recent material from de Villemereuil (2018), I think it's best to cite this most recent reference:

> P de Villemereuil (2018). Quantitative genetic methods depending on the nature of the phenotypic trait. *Annals of the New York Academy of Sciences.* The Year in Evolutionary Biology 1422, 29–47

1 What is heritability?

1.1 Definition

Let's study a phenotypic trait in a given wild population. This trait varies among individuals, each having a more or less different value from the others. This variation is quantified by a variance (called *phenotypic variance* or V_{P}). It originates from different sources, but in a schematic way, we will state it has a genetic component V_{G} and an environmental component V_{E}:

$$V_{\mathrm{P}} = V_{\mathrm{G}} + V_{\mathrm{E}} \tag{1}$$

The genetic variance itself originates from different sources. To keep it simple, and following the principles introduced by Fisher (1918), we will assume an *additive* component V_{A} and a *non-additive* component V_{NA} (dominance, epistasis, etc.). We thus write:

$$V_{\mathrm{P}} = V_{\mathrm{A}} + V_{\mathrm{NA}} + V_{\mathrm{E}} \tag{2}$$

The interest of that variance decomposition lies in the fact that V_{A} stands for the part of phenotypic variability which is actually genetically transmitted to the descendants (at the level of the whole population). The reasons for why only V_{A} is transmitted are linked to how the "additivity" is constructed and outside the scope of this tutorial (for a detailed explanation, the reader can read Lynch & Walsh 1998, pp. 65-79). For natural selection to result in evolutionary change, a part of the phenotypic variability, on which it acts, must be "transmittable" (we say that the trait is *heritable*).

Thus we introduce a quantity, called *heritability*, which allow us to measure how much the phenotypic variability of a trait in a given population is likely to be transmitted to the offspring. The heritability h^2 is here defined as the contribution of additive variance into the variability of the phenotype. Using previous notations :

$$h^2 = \frac{V_{\mathrm{A}}}{V_{\mathrm{P}}} \tag{3}$$

The heritability has a value that lies between 0 and 1.

• 1.2 A few words

- Heritability is *not* heredity! Heredity is the transmission of a phenotypic value from a parent to his offspring, while heritability is the transmission of the phenotypic variability within a population from generation to generation. For example, the number of legs in humans has a no heritability (any variation would be environmental), but is totally hereditary.
- It is important to stress that the heritability of a trait is defined for a given population at a given time. This quantity can vary between populations, and from time to time.
- Heritability can be a poor measure of additive genetic variance across contexts (e.g. labs vs. wild) and species, especially in cases were the environmental variance is likely to dominate as a source of variation.

• 1.3 Possible biases on heritability

A bias on heritability can originates from confouding resemblance between individuals, which could be misinterpreted as additive genetic effects. We thus need to get rid of two main sources of nuisance: the rest of the genetic effects (link to a correct estimation of V_{NA}) and the environmental effect.

▸ *1.3.1 Genetic nuisance*

They are mainly linked to dominance effects and epistasis, which, if they are badly accounted for, can lead to an overestimation of heritability. Hopefully, some methods are not sensitive to dominance effects (see subsection 1.4). Epistasis is a more complex and generalised issue, but it is a second order effect which is generally neglected by the models. Inbreeding can also lead to an overestimation of heritability, if not accounted for. There is also some factors that are hard to take into account, such as linkage…

1.3.2 Environmental nuisance

They are of various type, yielding greater resemblance between individuals than expected based on relatedness. Among others:

- **Common environment** Related individuals in a common environment (birth spot, habitat, etc.) are likely to show extra phenotypic resemblance.
- **Parental effects** Some characters of parents (quality, immunity, etc.) can yield an extra resemblance between their offspring and them; or more generally, extra resemblance within their offspring, independently of their phenotypes[1].
- **Assortative mating** A tendency of individuals to mate with partners sharing the same phenotype will lead to extra resemblance within their offspring.

From a practical point of view, the sensitivity to genetic effects (mostly dominance and inbreeding) will depend on the approach used to measure heritability. Environmental effects can sometimes be avoided with a careful design. The animal model approach allows to integrate some of theses effects in the model.

1.4 Which approach to infer heritability in the wild?

Different approaches can be used to infer heritability in the wild. They are characterised by both the type of data (especially, the type of relatedness between individuals available in the data) and the kind of statistical methods used.

Sibling design This approach uses the relatedness between (full- or half-)siblings in order to measure heritability. An ANOVA model compares the within-family to the inter-family sum-of-squares to estimate heritability. Because of an experimental protocol difficult to apply in wild population and an annoying issues with biased estimation of the heritability due to dominance and common environment effects (see above), this approach is less used in wild animal population. It is however important to note that the designs using half-sibling are not sensitive to dominance effects, but they do require some methodological conditions (certain identification of both parents, polygamy or sequential monogamy, etc.).

Parent-offspring regression This approach uses the regression of the phenotype of the mid-parent (or one of the parents) on the mean phenotype of the offspring. Well studied, its properties are well-known (Falconer & Mackay 1996; Roff 1997; Lynch & Walsh 1998). This approach is not biased by dominance effects, but ignore issues linked to inbreeding. More over, although a correction exists concerning assortative mating (Falconer & Mackay 1996, p.178), the parent-offspring regression remains sensitive to common environment between parents and offspring (territory inheritance, weak dispersal, etc.) and to *transgenerationnal* parental effects (i.e. parental effects which depend on the phenotype of the parent).

[1]The difference is important, since the former kind is what most biologists have in mind, whereas only the last kind can be modeled using the animal model, see subsection 2.3.

Animal model The animal model is a more complex approach, in the sense that it doesn't use only one kind of relatedness, but uses every known relationship in the whole pedigree of the population. It thus uses the maximum of information available, and to take into account inbreeding. It is a mixed model and thus can take into account several factors (mostly environmental) in order to avoid biases described in subsection 1.3. We will focus on this approach in the following parts.

2 The animal model in theory

2.1 Basic principle

The animal model uses a pedigree of the wild population. Such a pedigree indicate the father (or sire) and mother (or dam) for each individuals, like in the following array:

```
   individual  mother father
1          A1     NA     NA     #NA stands for Not Available value
2          A2     NA     NA
3          A3     NA     NA
4          A4     NA     NA
5          A5     A2     A4
6          A6     A1     A3
7          A7     A1     A3
8          A8     A1     A3
9          A9     A2     A4
10        A10     A2     A4
```

The individuals for which both the father and the mother are unknown are often called *founders* or *base population*. They generally are individuals from the beginning of the survey or immigrants. The individuals from the base population are assumed unrelated to each others. Note that this means that such *base population* can be composed, in majority, of immigrants, making it a weird reference. Accounting for immigration groups can be performed with some work using the animal model framework (Wolak & Reid 2017). This kind of pedigree allows the calculation of relatedness among all individuals of the population:

```
     A1   A2   A3   A4   A5   A6   A7   A8   A9   A10
A1     1    0    0    0    0  1/2  1/2  1/2    0    0
A2     0    1    0    0  1/2    0    0    0  1/2  1/2
A3     0    0    1    0    0  1/2  1/2  1/2    0    0
A4     0    0    0    1  1/2    0    0    0  1/2  1/2
A5     0  1/2    0  1/2    1    0    0    0  1/2  1/2
A6   1/2    0  1/2    0    0    1  1/2  1/2    0    0
A7   1/2    0  1/2    0    0  1/2    1  1/2    0    0
A8   1/2    0  1/2    0    0  1/2  1/2    1    0    0
A9     0  1/2    0  1/2  1/2    0    0    0    1  1/2
A10    0  1/2    0  1/2  1/2    0    0    0  1/2    1
```

Note

Due to the influence of inbreeding, the diagonal elements are often (slightly) larger than 1 and they are expected to be exactly equal to 1 only for the founders. This matrix is the additive genetic relatedness matrix $\boldsymbol{A}$. It is included in a random or mixed model in order to estimate its associated variance component V_{A} (i.e. additive genetic variance).

• 2.2 Model description

Let's have several measures of a phenotype y on n individuals, during a given period and in a given wild population. These measures are gathered in a vector:

$$Y = \begin{pmatrix} y_1 \\ \vdots \\ y_n \end{pmatrix}. \tag{4}$$

These data are given along a pedigree of the population containing information for all n measured individuals. Our aim is to separate the variation in the phenotypic trait Y between an additive genetic variance V_{A} and the "rest" (often too quickly assimilated to environmental variance). In order to do that, we will consider the phenotype y_i of the individual i as a variation around the average phenotype μ in the base population (corresponding to the founders). We will separate the variation originating from the transmissible additive genetic effects a_i and the "rest", wich we will call the residual and note e_i:

$$y_i = \mu + a_i + e_i. \tag{5}$$

We still need to define the distribution of the breeding values a_i and residuals e_i (μ is a simple constant). The breeding values are assumed normally distributed, given the pedigree, and thus the additive genetic matrix $\boldsymbol{A}$:

$$\begin{pmatrix} a_1 \\ \vdots \\ a_n \end{pmatrix} \sim \mathcal{N}(0, \boldsymbol{A}V_{\mathrm{A}}) \tag{6}$$

V_{A} is the additive genetic variance we are looking to estimate here, in order to estimate the heritability of the trait. The residuals e_i are also normally distributed:

$$\begin{pmatrix} e_1 \\ \vdots \\ e_n \end{pmatrix} \sim \mathcal{N}(0, \boldsymbol{I}V_R) \tag{7}$$

where $\boldsymbol{I}$ stands for the identity matrix and V_R is the residual variance.

From Equation 5, Equation 6 and Equation 7, the key assumptions of the animal model are:

- The breeding values a_i are normally distributed and correlated among related individuals. The function of the pedigree is to structure the correlation between individuals by taking into account their relatedness.
- The residuals e_i are normally distributed and uncorrelated. They also are independent from the breeding values (e.g. no environment-genotype covariation).
- As a result, the Y trait can be considered a "Gaussian trait", i.e. a trait which can be analysed in a way so that the residuals e_i are indeed normally distributed, even it requires the inclusion of some fixed effects to account for parts of the distribution of Y.

The main outputs of the animal model are an estimate of the additive variance $\hat{V}_A$ and an estimate of the residual variance $\hat{V}_R$. These two variance components sum to the total phenotypic variance V_{P}. We thus estimate the heritability as:

$$\hat{h}^2 = \frac{\hat{V}_A}{\hat{V}_A + \hat{V}_R} \tag{8}$$

2.3 Going further into the model

2.3.1 Adding random effects

In order to account for some possible biases (common environment, parental effects…), it is possible to add random factors in the model. When those random factors ($U_1, \ldots, U_K$) are added, they have to be taken into account for the calculation of the total phenotypic variance, such as:

$$\hat{h}^2 = \frac{\hat{V}_A}{\hat{V}_A + \hat{V}_{U_1} + \ldots + \hat{V}_{U_K} + \hat{V}_R} \tag{9}$$

Note Sometimes, it can also be relevant to exclude some factors from the denominator, so that the computed heritability is *conditional* to those factors, i.e. computed as if those factors were kept constant.

Parental effects Including the identity of the parents (or only one parent) as a random effect allows to account for possible parental effects. However, it has to be stressed that, included that way, parental effects only take into account the resemblance among siblings from the same parents, but in no way the resemblance between parents and offspring! For example, one famous maternal effect is the quality of mother's milk in mammals: mothers of the best phenotypic "quality" are supposed to give the best milk, which in turns help her offspring to grow healthy. In this context, including the identity of the mother into the model will state that the offspring of a same mother are likely to develop a close phenotype because of the more or less good quality of the mother's milk, but it doesn't imply any resemblance between the 'quality' of the mother and her offspring. In other words, the parental effect is independent from the parental phenotype (this a non transgenerationnal effect)! A kind of parental effects in animal models that are transgenerationnal are *genetic* parental effects (see Wilson et al. 2010, for example). This effect assume a hypothetical 'maternal performance' trait which is a summary of potential maternal traits acting on the offspring's phenotype and is heritable. Including a fully parametrized parental effect (including measured maternal traits acting on offspring's phenotype) is not straightforward using animal model, but a more general type of models exists that allow for this (Kirkpatrick & Lande 1989; Räsänen & Kruuk 2007; Day & Bonduriansky 2011).

Dominance effects Dominance effects are problematic only through a supplementary correlation among full-sibs. Since the animal model uses all kind of relationships, it is sensitive to dominance effects, especially if the pedigree contains large siblings groups. A dominance matrix is not necessarily difficult to construct, but the calculation has to be made before the use of MCMCglmm (see subsection 3.8).

Repeated measures If repeated measures on individuals are available, it is possible to estimate what is called a *permanent environment* effect, by including the identity of individuals as a random effect. It is important to model such permanent environment, because without this, the animal model would consider repeated measures as arising from clones, thus biasing the additive genetic variance estimate. The permanent environment effect also accounts for a part of the non-additive genetic effects (but doesn't account for the whole dominance effect for example!). It is important to realize that the total phenotypic variances V_P are not comparable between a design using repeated measures and mean value for each individual (taking the mean reduces V_P).

2.3.2 Adding fixed factors

Including fixed effects The animal model is in essence a mixed model, allowing also for fixed effects. It can be interesting to add fixed factors to the model, in order to account for some biological or design-related issues likely to bias our heritability estimation. But, it is important to note that fixed effects also participate to the total phenotypic variance (Wilson 2008), and thus should be included in the denominator of the heritability. As an example, a model including a fixed effect based on a predictor x could be written by extending Equation 5:

$$y_i = \mu + bx_i + a_i + e_i. \tag{10}$$

where b is the slope, for which we will note the estimator $\hat{b}$.

Fixed-effect variance The part of the variance contributed by the fixed effects (V_{F}) is akin to what measures the numerator of the famous coefficient of determination R^2 and can be defined as the variance of the predicted values of from the fixed effects (noted $\hat{y}$, see de Villemereuil et al. 2018):

$$V_{\text{F}} = V(\hat{y}_i) \tag{11}$$

In our example above, we have $\hat{y}_i = \hat{b}x_i$ and so $\hat{V}_{\text{F}} = V(\hat{b}x_i)$. We will see below an example of how this quantity can be computed from a MCMCglmm output.

NOTE

Note that an important hypothesis behind the calculation in Equation 11 is that the standard error in the estimation of $\hat{b}$ is negligible (and thus produce a negligible bias) in front of the value of V_{F}. In classical animal models, the sample size of the data is generally large enough, hence the standard error in fixed effects estimates is low enough, so that the computation above is relevant.

Computing the heritability with fixed effects Once $\hat{V}_{\text{F}}$ has been computed, the calculation of h^2 is straightforward, as we simply need to add it to the denominator:

$$\hat{h}^2 = \frac{\hat{V}_A}{\hat{V}_A + \hat{V}_{\text{F}} + \hat{V}_R} \tag{12}$$

NOTE

Just as we noted in for the random effects, some part of the fixed effects could be removed from the computation of $\hat{V}_{\text{F}}$ to compute a *conditional* heritability, i.e. based on the assumption that those parameters are kept constant. It is a slightly more complicated matter, however, as fixed effect predictors could be correlated between them and it could also beg the question as to *which* value they should be kept constant. More information on this issue can be found in de Villemereuil et al. (2018).

2.4 Pros and cons

More statistical power, more flexibility... Using the entire pedigree of the population, the animal model has better resources toward a precise estimation of the heritability than the parent-offspring regression. It is also more accurate by taking into account inbreeding and any selection event occurring "since" the founders. Adding random effects also allows for explicitly model dominance effects, common environment effects (nest, habitat, year...) or (non transgenerationnal) parental effects. A multivariate variant is also available, yielding genetic covariance between several traits.

...but everything is not perfect! The fact that the animal model uses the whole range of relationship in the population is its strength, but also its weakness. For example, the parent-offspring regression is only based on "vertical" relationships between individuals and thus ignores any non transgenerationnal effects: a dominance effect or a non transgenerationnal parental effect will induce no bias on heritability estimation. As a consequence, it is highly important to properly think about all different sources of effects likely to induce a bias on heritability and (if the data structure is sufficient) to indicate them to the model using fixed or random effects. It is also important not to neglect simpler approaches such a parent-offspring regression (or half-sibs design if possible), at least as a checking step!

2.5 Generalised animal model

2.5.1 Theory of GLMMs

It happens often that phenotypic traits cannot be modelled by a normally distributed random error. This is especially the case for count, categorical or binary data. In such cases, one has to rely on Generalised Linear Mixed Models (GLMM) rather than LMM. GLMM allows for the use of many different kind of distributions by using a hierarchical structure going from a normally distributed (hypothetical) latent trait to the observed data.

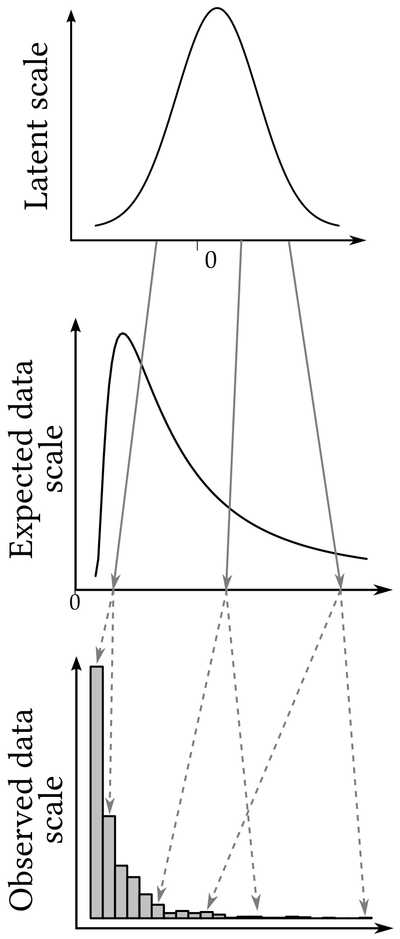

Figure 1: The three scales of the Generalised Linear Mixed Model (here using Poisson with a log link function as an example). The error terms are normally distributed on the latent scale, but follows a Poisson distribution on the observed data scale (conditionnally on the latent scale).

This structure consists of three "scales" depicted in Figure 1 and which can be written using the following equations, e.g. using a Poisson with a log-link model:

$$\ell_i = \mu + a_i + o_i, \tag{13a}$$

$$\eta_i = \exp(\ell_i), \tag{13b}$$

$$y_i \sim \mathcal{P}(\eta_i), \tag{13c}$$

where it $\mathcal{P}$ stands for the Poisson distribution. It should be noted that the exponential is the inverse of the logarithm function, which is how it comes into play in Equation 13b.

In Equation 13a, ℓ refers to something called the *latent scale* (see Figure 1). By comparing it to Equation 5, we can see that the same assumptions are made for ℓ as in any LMM. To reflect the fact that the "error" on the latent trait is not the residual error of the model, we changed the notation of the residual e to o. The term o is still normally distributed and stands for the additive over-dispersion of the model (Nakagawa & Schielzeth 2010). Accordingly, we will note the variance associated to o as V_{O}.

In de Villemereuil et al. (2016), Equation 13b is said to refer to the "expected data scale". This is because the term η is the individual expectation around which the observed data are realised (see Figure 1). The transition from the latent scale to the expected data scale is performed by the inverse of the link-function (here, inverse of the logarithm, which is exponential). The link function is "mapping" the variations on the latent scale to variations compatible for the distribution used. For example, whereas the latent trait varies between $-\infty$ to ∞, a Poisson distribution can only use positive values, hence the use of the exponential function which match these input and output realms (the exponential of something cannot be negative).

Finally, Eq. 13c models the "observed data scale" by adding, around the expectation η, an error term from a Poisson distribution (here noted $\mathcal{P}$). This is the part that models (and thus should fit) the actual data Y.

Important

Most (and very often, all) parameters **are inferred on the latent scale**, and **not** on the observed data scale. All parameters commonly interpreted in quantitative genetics (population mean μ, additive genetic variance $V_{\mathrm{A},\ell}$ and all other variance components) are thus related to a hypothetical latent trait, and hence **not directly to the phenotypic trait of interest** (see more about this in section 2.5.2).

2.5.2 Quantitative genetics and GLMM

How is the use of GLMM justified for quantitative genetics analysis? What should we do differently than what we are doing when using plain LMM? Many people think that GLMM are just "LMM with a different distribution", but we just saw that the reality is more complex than that, especially because the model has now three different scales, each with a particular behaviour. We thus need a framework to *travel* from a scale to another. This is the purpose of the QGglmm package that we will use the tutorial below. But, it is worth mentioning a few details on quantitative genetics and GLMMs before starting the pratical part of this tutorial.

Note

Most of the concepts explained in this theoretical part, as well as more practical considerations when using the QGglmm package are available in the vignette of the package: https://cran.r-project.org/web/packages/QGglmm/vignettes/QGglmmHowTo.pdf

The first thing to mention is why we need the latent trait to be Gaussian (or a latent trait at all...). This stems from models underlying quantitative genetics, especially Fisher (1918)'s infinitesimal model: the results of a large number of additive effects, each with a small effect, will result in a normally distributed genetic component. A similar line of reasoning (e.g. using the central-limit theorem) allows us to assume the same kind of distribution for the environmental effects. Hence, more than justified, it is *needed*[2] that at some point, *something* is normally distributed. We simply call this something "latent trait".

A second thing to mention is that GLMM are in essence *very noisy* models. There are three main sources of noise. A first source is the latter part of the model (Equation 13c) in which observed values are drawn around the expected values following the error distribution (e.g. a Poisson or a binomial distribution). This is the actual "error process" of the model. One thing is very important to note with this source of noise: contrary to the normally-distributed noise of a LMM, the level of noise almost always depends on the actual expected value. Indeed, the variance of a Poisson is equal to the mean, the variance of a binomial distribution depends only on the mean, etc... This means that we assume that a part of the phenotypic variance is irreducible: there's always going to be some variance, depending on the value of the trait.
The second source is this link-function. It is not *creating* noise (it's a function, not a statistical

[2]Well, obviously, this is assuming that the infinitesimal model is a good approximation of the reality.

distribution after all), but it can *amplify* the noise from the latent scale to a great extent. Think about the Poisson-log model above: we take the exponential of the latent scale. This means that values that are close on the latent scale, say 1, 2 and 3, will give respectively 2.7, 7.4 and 20 on the expected data scale: large values become even larger!
Finally, the last source of noise is the over-dispersion variance present in the latent scale. Despite GLMMs being somewhat noisy already, it is frequent that this variance is required for a good model fit[3].
Why is it important that GLMM are noisy in the context of quantitative genetics? Well, this means that phenotypic variance on the observed data scale tends to be large, especially when compared to the original variance on the latent scale. The direct consequence of this is that heritabilities inferred on the observed data scale are expected to be *rather small*! For example, because GLMMs always assume environmental noise from the error distribution, heritability on the observed data scale can never reach the maximum value of 1.

A third and last thing to mention is something quite important from a quantitative genetics perspective: the link-function is almost never linear! Why is that important? Because it breaks the additivity of the genetic effects to some degree. Even if only additive effects are assumed on the latent scale, the result on the observed data scale is a mix of additive and non-additive effects (simply because it went through the inverse-link-function). When computing the heritability, we want to extract only the additive part, which is all what the package QGglmm introduced below is about. But this has further consequences. Narrow-sense heritability is nothing like repeatability[4] when they are computed on the observed data scale. This means that some of the computations and advice available in Nakagawa & Schielzeth (2010) regarding repeatabilities are not applicable to narrow-sense heritability on the observed scale. However, broad-sense heritability (i.e. including the non-additive effects) can be computed as a repeatability-like estimate (a.k.a. intra-class correlation coefficients, ICC).

Note

QGglmm also provides a way to compute ICCs, even for non-genetic components, see the package's vignette.

2.6 Multi-trait animal model

2.6.1 Multiple traits and their covariance

A multi-trait animal model is a multivariate animal model in which the response variable is "extended" to include a series of different traits (possibly of different error distributions, if we are considered a multi-trait generalised animal model). It allows the analysis of several phenotypic traits at once. But the main interest of such model is not to compact an analysis of different traits in a single run[5]. The main interest is that it allows inference about the *relationship*, especially of genetic origin, between traits.

One interesting measure of such relationship is the genetic and residual covariance between traits. The genetic covariance (or correlation) between two traits informs how much of their genetic basis is shared (e.g. through linkage and pleiotropy). For a maximal covariance (correlation of 1), the two traits entirely share the genes that influence their variance. A genetic modification on a trait would translate into the same amount of modification on the other. On the contrary, a null

[3]Sometimes, the over-dispersion variance is also required for the method to work: it is the case for MCMCglmm

[4]Heritability is sometimes considered as an "additive genetic repeatability" and they share most of their features when assuming a LMM. Especially, they are both some kind of statistics called intra-class correlation coefficients.

[5]It would be more time efficient to run each analysis in parallel.

covariance means that the traits are genetically independent[6], meaning it is possible to genetically modify one without impacting the other. The residual covariance can be interpreted in roughly the same way, except it is supposed to capture everything that was not modelled (hence "residual"), but still constitute a relationship between the two traits (typically, environmental sources of covariance between the two traits).

As an example, let's consider you are a breeder and you want to maximise, both at the same time, the output of milk and meat yield of your cattle. So, you start selecting for both at the same time. At some point, however, it is not possible to select for further improvement of both milk and meat yields, although you realise you could increase either milk or meat yields separately. Why is this? A simple explanation would be that milk and meat yields are negatively genetically correlated, for the simple reason that any genetic modification that change metabolism allocation to either meat or milk production is going to negatively affect the other, as the total energetic budget is limited[7]. Having the knowledge that such negative genetic correlation exists can help a lot in such cases. In general, it informs us on the existence of such evolutionary trade-offs. Note that it could well be that the residual (environmental) covariance between the two traits is positive here, as variations in the environment of the cattle (e.g. quality of food) will have an impact on both milk and meat yields.

2.6.2 How does it work?

Basically the multi-trait model "stacks" the two traits (say Y_{T_1} and Y_{T_2}) into a very long vector:

$$\begin{pmatrix} \boldsymbol{Y}_{T_1} \\ \boldsymbol{Y}_{T_2} \end{pmatrix} = \begin{pmatrix} y_{1,T_1} \\ \vdots \\ y_{n,T_1} \\ y_{1,T_2} \\ \vdots \\ y_{n,T_2} \end{pmatrix} \tag{14}$$

Then it uses a model similar to Equation 5 (with adapted notations and, often, an intercept that depends on the trait). The distribution of the random is likewise "stacked" and the effect variance is replaced by a variance-covariance matrix. For the additive genetic effect, it thus becomes:

$$\begin{pmatrix} \boldsymbol{a}_{T_1} \\ \boldsymbol{a}_{T_2} \end{pmatrix} = \begin{pmatrix} a_{1,T_1} \\ \vdots \\ a_{n,T_1} \\ a_{1,T_2} \\ \vdots \\ a_{n,T_2} \end{pmatrix} \sim \mathcal{N}\left(\begin{pmatrix} 0 \\ 0 \end{pmatrix}, \boldsymbol{G} \otimes \boldsymbol{A}\right) \tag{15}$$

where $\boldsymbol{G}$ is the additive genetic variance-covariance matrix, which contains the additive genetic variances in the diagonal and additive genetic covariances on the off-diagonal elements:

$$\boldsymbol{G} = \begin{pmatrix} V_{\mathrm{A},T_1} & C_{\mathrm{A}} \\ C_{\mathrm{A}} & V_{\mathrm{A},T_2} \end{pmatrix} \tag{16}$$

The operation $\otimes$ is a Kronecker product, useful here to condense the notation. Basically, it multiply both matrix "by blocks" as follows:

$$\boldsymbol{G} \otimes \boldsymbol{A} = \begin{pmatrix} V_{\mathrm{A},T_1}\boldsymbol{A} & C_{\mathrm{A}}\boldsymbol{A} \\ C_{\mathrm{A}}\boldsymbol{A} & V_{\mathrm{A},T_2}\boldsymbol{A} \end{pmatrix} \tag{17}$$

[6]I know, I know... Absence of correlation does not mean independence. But it works here because of the assumptions that the two traits are jointly normal, at least on the latent scale for non-Gaussian traits.

[7]I realise this is a very simplified explanation...

Note that $\boldsymbol{G} \otimes \boldsymbol{A}$ is a very big matrix (the number of traits times the number of individuals)! The residual component is designed exactly the same way:

$$\begin{pmatrix} \boldsymbol{e}_{T_1} \\ \boldsymbol{e}_{T_2} \end{pmatrix} = \begin{pmatrix} e_{1,T_1} \\ \vdots \\ e_{n,T_1} \\ e_{1,T_2} \\ \vdots \\ e_{n,T_2} \end{pmatrix} \sim \mathcal{N}\left(\begin{pmatrix} 0 \\ 0 \end{pmatrix}, \boldsymbol{R} \otimes \boldsymbol{I} \right) \tag{18}$$

NOTE

If these equations were a bit too much for you, just keep in mind: the aim of such multi-trait models is to infer (among other things) the variance-covariance matrices $\boldsymbol{G}$ and $\boldsymbol{R}$ (which dimensions are the number of traits), rather than just the variances V_A and V_R, as we've been doing for now.

3 The animal model in practice using MCMCglmm

3.1 Some notions about Bayesian statistics

This section does not aim to be a lecture about Bayesian inference. However, at least some basics are needed for a proper use of the MCMCglmm package. The output of a Bayesian inference is a *posterior distribution*, i.e. a probabilistic distribution associating each value of a parameter to a probability (or degree of belief). The inference model is made of a *likelihood* function (in everything identical to the "classical" frequentist counterpart) and a *prior distribution* of the parameter(s) to be estimated. The likelihood model has been described in the previous section, we just need to choose the prior distributions for the parameters to be estimated.

NOTE

For more information about Bayesian statistics and MCMCglmm, I advise the reader to consult Jarrod Hadfield's course notes, who developed the MCMCglmm package.

What is the MCMC estimation method? The aim of the MCMC algorithm (Markov Chain Monte Carlo) is to numerically approximate the posterior distribution of the parameters. To do so it uses an algorithm based on consecutively drawing a new value for a parameter, based on the value of the other parameters. After a *convergence* phase (often rather small in MCMCglmm), the MCMC algorithm tends to propose values within the posterior distribution of the parameters. Saving the value of the parameter at each iteration (or a subset, see later), we get a series of values drawing the posterior distribution of interest (just as a series of normally distributed values draw the famous bell curve). The whole suite of generated values are referred to as *chains* (because they are realisations of a Markov Chain process). In order to get a good picture of this posterior distribution, we would ideally need a rather large sample of that chain (say above 1000), for which the *autocorrelation* is negligible. Indeed, the successive iterations of the MCMC algorithm have the annoying tendency to be correlated from one draw to the next. This is due to the fact that the proposal of a new value is based on its current value, and those of the other parameters. This *autocorrelation* reduce the *effective size* of our sample. The *effective size* of a draws sample is the size of an ideal uncorrelated sample (all draws are independent) equivalent to our draws sample, in terms of how much information we obtained regarding the posterior distribution. For example, 10,000 draws highly correlated might have an effective size of 100 (i.e. they are equivalent to 100 independent draws), since they are mostly redundant in the information about the posterior distribution they convey. Since the iterations are correlated with those in proximity, we can pick up in advance a *thinning interval* which reduces the final draws sample size (for example, we choose to keep only one iteration value every 10 iterations). This allows to spare memory and lightens further analysis. To put it in a nut shell, there are two important issues to monitor when using the MCMC:

- **The convergence** It is important to check we waited long enough for the convergence (this waiting period is often called the *burn-in*) to actually happen, before saving iteration values. Unfortunately, there is no way to tell in advance how long the burn-in has to be, so *post hoc* checks are generally used.
- **The autocorrelation** The level of autocorrelation of a MCMC chain highly depends on the model we are fitting, as well as on the data. Monitoring the final autocorrelation of our draws sample, and more importantly, the effective sample size of the parameters is thus very important to ensure we obtained enough information about the posterior distribution of such parameters.

Important

It is quite important to realise that autocorrelation is nothing absolute, and depends on both the model and the data. Often, authors report the total number of iterations, length of the burnin period and thinning interval: this is actually meaningless without further information on the autocorrelation, as a reader/reviewer cannot assess whether the MCMC was long enough based on this information, even with expert knowledge. Reporting the effective sample size (minimal or for all parameters) is, for example, much more informative than reporting the thinning interval used! As mentionned above, thinning is for computational convenience and does not help reduce autocorrelation in itself, only running MCMC for longer does.

3.2 Choosing a prior distribution

3.2.1 What is a good prior?

The best prior is a prior that reflects the prior degree of belief of the researcher before starting the analysis (e.g. reflecting the state-of-the-art). In practice, priors are scarcely used that way because *(i)* people tend to not be at ease with the notion of "prior degree of belief" in scientific research[8] and *(ii)* such informed priors can be difficult to model mathematically. As a result, most people prefer to resort to *non informative* priors, which means that the prior should not influence the estimated posterior distribution. In most cases, 'flat' priors check this criterion, but it is not a golden rule. It is actually quite difficult, without a prior sensitivity study, to predict the influence of the prior. Worse, the notion of non informative prior for a variance component is almost impossible to be defined relevantly (Gelman 2006). Fortunately, the strength of the prior fades away whith the sample size of the data: with sufficient sample size, this prior issue becomes negligible. For the definition of priors, the MCMCglmm package has specific distributions already implemented.

3.2.2 Prior distribution for a random effect variance

Regarding the prior distribution of variances, MCMCglmm uses an inverse-Gamma distribution, which is a common choice. In the package, the distribution is parametrized by two parameters `nu`

[8]Whether this attitude is justified or not is subjected to heated debates, and is not the point of this tutorial.

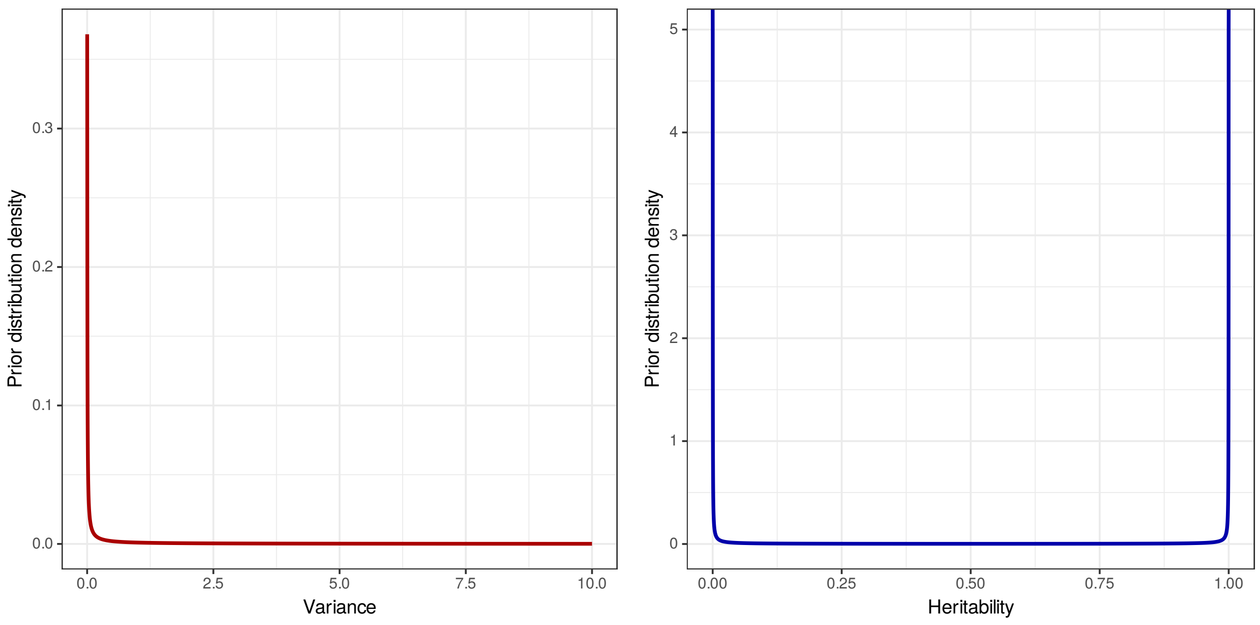

Figure 2: Shape of inverse-Gamma$(0.001; 0.001)$ prior distribution on variance components (left) and induced shape of the prior for heritability (right).

and V[9]. A possible set of parameters would be $\mathrm{nu} = 0.002$ and $\mathrm{V} = 1$ (Figure 2). Indeed, it allows for a weakly informative prior on variance components and U-shaped prior (with a very steep shape on the borders in 0 and 1) on the heritability. This choice is obviously not the only one, but it has the quality of being 'classical' (though not appropriate for too small variances, see Gelman 2006) and generally weakly informative. To define this prior on variances using MCMCglmm, we use an R list as follows:

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

In this list, the `R` argument stand for the prior on the residual variance. The `G` argument (itself a list) is for random effects variance (called `G1`, `G2`, etc.). In presence of 3 random effects in the model, we need to define 3 priors in `G`:

```r
prior <- list(R = list(V = 1, nu = 0.002),
              G = list(G1 = list(V = 1, nu = 0.002),
                       G2 = list(V = 1, nu = 0.002),
                       G3 = list(V = 1, nu = 0.002)))
```

3.2.3 Parameter expanded prior distribution

A variation of the above implemented in `MCMCglmm` a *parameter expanded* prior (Gelman 2006), which allows for a bit more flexibility and usually results in better mixing of the chains. The idea behind the parameter extensions lies into splitting the random effect u_i into two independent components:

$$\begin{gathered} u_i = \alpha\, \eta_i \\ \text{with } \eta_i \sim \mathcal{N}(0; V_\eta) \end{gathered} \tag{19}$$

[9]On the Wikipedia website, the inverse Gamma is parametrized differently using α and β notations. Since MCMCglmm notations are not usual, here are their 'translation': $\alpha = \frac{\mathrm{nu}}{2}$ and $\beta = \frac{\mathrm{nu} \times \mathrm{V}}{2}$. Thus, a distribution parametrized by $\mathrm{nu} = 0.002$ and $\mathrm{V} = 1$ is actually an inverse Gamma$(0.001; 0.001)$.

Doing so, we implicitly split the variance of the random effect u in two: $V_u = \alpha^2 V_\eta$. We have the following prior distributions for α and V_η:

$$\begin{aligned} \alpha &\sim \mathcal{N}(0; V_\alpha) \\ V_\eta &\sim \text{Inverse-Gamma}(\text{V}; \text{nu}) \end{aligned} \tag{20}$$

OK... This is a bit technical. What's important here is to remember that using this kind of prior might help in obtaining better mixing of chains, and can allow for a bit more flexibility in its shape.

So, how does one do this in MCMCglmm? We simply add more parameters in our prior statement. For example, an interesting prior is this one:

```
prior <- list(R = list(V = 1, nu = 1),
              G = list(G1 = list(V = 1, nu = 1, alpha.mu = 0, alpha.V = 1000)))
```

It defines a 'Fisher' prior[10] that is more informative toward small variances, but also puts less weight in very tiny values close to zero. It is usually a nice prior when variances are expected to be small. Note that the residual variance cannot have residual expanded parameters. In order to fit the "scenario" here, I've set `nu = 1` for this residual variance, which contributes to the MCMC mixing better when variances are small.

3.2.4 Prior distribution of fixed effects

Regarding the prior distribution of fixed effects, the package defaults to a very wide Normal distribution (i.e. with an extremely large variance), which is a relevant and quite consensual choice few users will have to customize further.

3.3 MCMCglmm( ) function parameters

3.3.1 The data

For the whole tutorial, we will use a (fake, of course) "phoenix" dataset that was first designed for de Villemereuil (2018). This dataset consists of the (ancient) monitoring of a population of phoenix birds, nesting in separate small woods, with natal dispersal, on which we measured the length of tarsus, weight, whether they were of white or golden plumage and the number of time they revived before eventually (and sadly) disappearing. Some more information like personality, and year, temperature and phase of the moon at the time of birth were also measured/recorded:

```
data <- readRDS("data.rds")
psych::headtail(data)
```

```
     animal    Wood Tarsus Weight Disp_Distance White Nb_Revival Personality Birth_Year
1         1 Wood_14  54.26 504.01         11.83     1          1      Medium       1535
2         2 Wood_10  47.47  483.1          5.96     1          0        Bold       1535
3         3 Wood_14  49.01 504.21           0.7     0          3      Medium       1535
4         4 Wood_16  51.07 481.94          2.87     0          0        Bold       1535
...     ...     ...    ...    ...           ...   ...        ...         ...        ...
997     997 Wood_13  48.39 496.33          1.69     0          1         Shy       1542
998     998 Wood_20  52.58 523.43         20.03     1          2         Shy       1542
999     999 Wood_15  45.04 472.84          6.94     0          0        Bold       1542
1000   1000 Wood_30  49.71 501.81          1.24     0          0      Medium       1542
```

[10]The distribution for G1 is a scaled F-distribution.

```
     Birth_Temp Birth_Moon
1         21.87       Full
2         17.34       Full
3         14.18      First
4         20.93        New
...         ...        ...
997       21.36        New
998       26.07       Full
999       19.17        New
1000      17.79       Last
```

Of course, a pedigree of the population is available.

3.3.2 Setting up the pedigree

For `MCMCglmm()` to work as an animal model, we indeed need to provide it with a data frame describing the population pedigree of that kind:

```
pedigree <- readRDS("pedigree.rds")
psych::headTail(pedigree)
```

```
      animal sire  dam
1          1 <NA> <NA>
2          2 <NA> <NA>
3          3 <NA> <NA>
4          4 <NA> <NA>
...      ...  ...  ...
997      997  766  860
998      998  787  840
999      999  795  799
1000    1000  794  852
```

The first column has to be named `animal`. The founders (whose father and mother are unknown) must be placed of the top of the array, because (more generally) a reproductive individual must appear before its offspring. MCMCglmm can accept a pedigree with missing parents (with `NA` in place of the parent's ID).

Note

Note the presence of the `animal` column in the `data` above. This column contains the individual IDs, which should correspond to the IDs in the `animal` column of the pedigree, although the ordering does not have to be the same and IDs in the pedigree can be missing in the data (but not the reverse!).

3.3.3 How to use `MCMCglmm()` function?

In order to fit a simple model, for example to analyse `Some_trait`, where we only estimate additive and residual variances, we can call the function likewise:

```
library(MCMCglmm)
prior <- list(R = list(V = 1, nu = 0.002),
              G = list(G1 = list(V = 1, nu = 0.002)))
model <- MCMCglmm(Some_trait ~ 1,
                  random    = ~ animal,
                    family   = "gaussian",
                    prior    = prior,
                    pedigree = pedigree,
                    data     = data,
                    nitt     = 100000,
                    burnin   = 10000,
                    thin     = 10)
```

The fist argument `Some_trait ~ 1` is a R formula giving the response variable (`Some_trait`, i.e. the phenotypic trait we want to analyse) according to some fixed effects (here none, we only specify an intercept with `~ 1`, which corresponds to our parameter μ in Equation 5). The argument `random = ~ animal` set the random effects: `animal` is a reserved variable to fit the additive genetic effect (in other words, we are stating the model we want to estimate V_A). The argument `family` set the distribution to use for the data (here `gaussian` for a normally distributed trait). The argument `prior` calls the list of parameters for prior distributions stored in the variable `prior`. Arguments `pedigree` and `data` speak for themselves. Finally, the three last arguments are to set up the properties of the MCMC: `nitt` is the total number of iterations, `burnin` is the number of iterations to drop at the beginning (convergence) and `thin` is the number of iterations stored in memory (here, one every ten iterations).

3.4 Results and diagnostic of the MCMC output

For this section, we will use “Tarsus length” as the phenotypic trait of interest.

3.4.1 Calling the function

The function is called likewise:

```
library(MCMCglmm)
prior <- list(R = list(V = 1, nu = 0.002),
              G = list(G1 = list(V = 1, nu = 0.002)))
model <- MCMCglmm(Tarsus ~ 1,
                  random   = ~ animal,
                  family   = "gaussian",
                  prior    = prior,
                  pedigree = pedigree,
                  data     = data,
                  nitt     = 100000,
                  burnin   = 10000,
                  thin     = 10)
```

```
MCMC iteration = 0
MCMC iteration = 1000
MCMC iteration = 2000
MCMC iteration = 3000
    ...          ...
MCMC iteration = 99000
MCMC iteration = 100000
```

Don’t forget to store the output of the function in a variable (here `model`), in order to have access to it afterwards, e.g.:

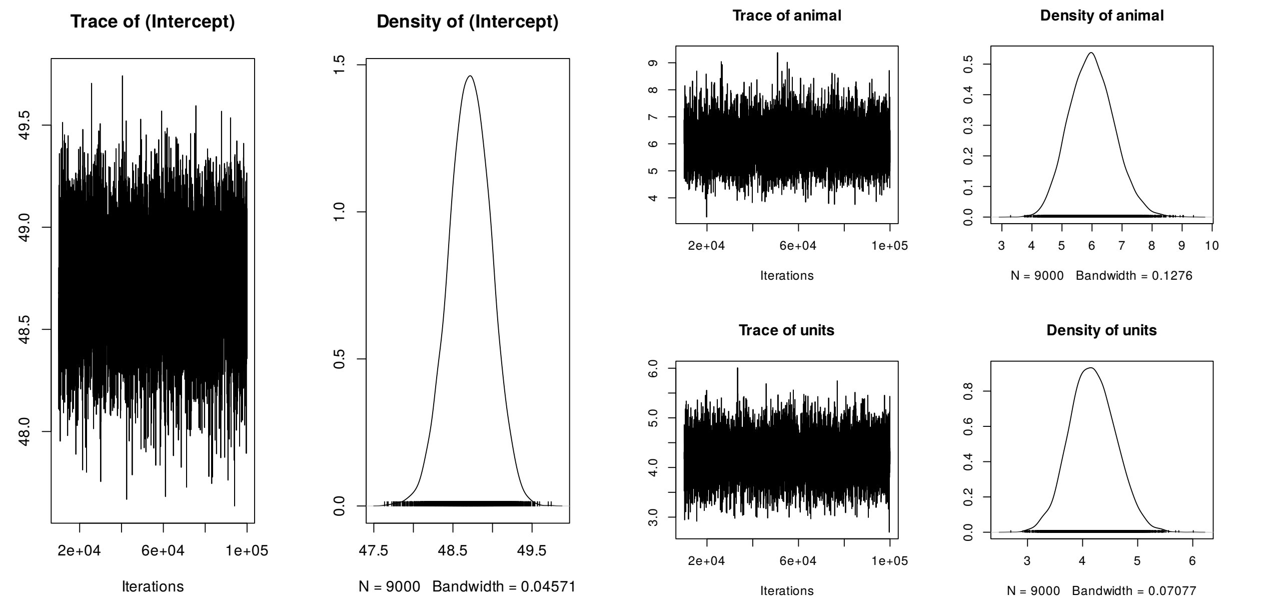

Figure 3: Trace of the intercept μ (or population mean, left) and the variances (right). The term `animal` refers to V_A and the term `units` refers to V_R.

```
saveRDS(model, "model_simple.rds")
```

The computation might take some time (even quite a long time), depending of the data, the parameters and, of course, the computer capacities.

3.4.2 Diagnostic of the MCMC

Before even looking at the estimates, the first thing to do is to check the behaviour of our MCMC algorithm. To do so, we need to focus on convergence and effective sample size of our draws samples. The output `model` has two main components, which are `model[["Sol"]]` and `model[["VCV"]]` (respectively for fixed effects and random effects variances). First of all, let's look at the "trace" of our chain (Figure 3):

```
plot(model[["Sol"]])
plot(model[["VCV"]])
```

Each of the three couples of graphs shows us the trace (left), i.e. the evolution of the sampled values along the iterations. It allows us to check the convergence (we should not see any trend in the trace) and that autocorrelation is weak (the values are widely spread, not following a traceable path). On the right of these graphs, we have an estimation of the posterior distribution for each component (`Intercept`, `animal` and `units`).

NOTE

If `MCMCglmm()` was run with the option `pr = TRUE`, then `"Sol"` also contains the predicted random effects (a.k.a. BLUPs) and thus can be very large!

From looking at Figure 3, there does not seem to be a lot of issue with auto-correlation, but we ought to have a more detailed look. The first and main thing to do is to compute the effective sample size from the our draws:

```
effectiveSize(model[["Sol"]])
```

```
(Intercept)
8158.747
```

```
effectiveSize(model[["VCV"]])
```

```
  animal    units
4485.220 4792.312
```

We see that the effective sample size of the mean (`Intercept`) is larger (and actually fairly close to the actual sample size) than the effective sample size for variance components. We need to recall how important is the effective size parameter: the aim of MCMC is to estimate the posterior distribution of the parameter of interest by sampling from it. To do so, we need the largest number of independent values as possible, which means a large effective sample size. In practice, an effective size above $1,000$ is recommended.

Let's see whether these differences in effective sample size can be explained by differences in auto-correlation (as they should):

```
autocorr.diag(model[["Sol"]])
            (Intercept)
Lag 0     1.0000000000
Lag 10    0.0161799116
Lag 50    0.0001262994
Lag 100 -0.0046778105
Lag 500 -0.0246060779
```

```
 autocorr.diag(model[["VCV"]])
                 animal         units
Lag 0     1.000000000   1.000000000
Lag 10    0.330839721   0.277521126
Lag 50   -0.004049377   0.009542044
Lag 100  -0.014149210  -0.009103030
Lag 500  -0.008306229   0.003328340
```

Here `Lag 10` stands for 'autocorrelation every 10 iteration values'. Since our `thin` parameter was 10, this refers actually to the correlation of every sampled value with the following one. We can see that there is little autocorrelation on the mean (`Intercept`). On the contrary, the autocorrelation on variance components becomes negligible rather with a lag of 50. The best way to decide whether this level of auto-correlation is problematic or not is rather to look at the effective sample size, whether it is acceptable or not (as draws samples from longer chains with higher autocorrelation can have a higher effective sample size than from a shorter chains with lesser autocorrelation).

Using MCMCglmm, the convergence is often quite fast. Here, it happens within the first hundred of iterations (Figure 4, look at the first regime of increasing values of `animal` in the first 100 iterations, and then the most stable regime after):

```
modelburnin <- MCMCglmm(Tarsus ~ 1,
                        random   = ~ animal,
                        family   = "gaussian",
                        prior    = prior,
                        pedigree = pedigree,
                        data     = data,
                        nitt     = 500,
                        burnin   = 1,
                        thin     = 1)
plot(modelburnin[["VCV"]])
```

For more complex models, the convergence might be much longer. Note that there are diagnostic tests of convergence, as the Heidelberg stationarity test (here, to bend the rules, the p-values must exceed 0.05 for stationarity to hold):

```
heidel.diag(model[["VCV"]])
```

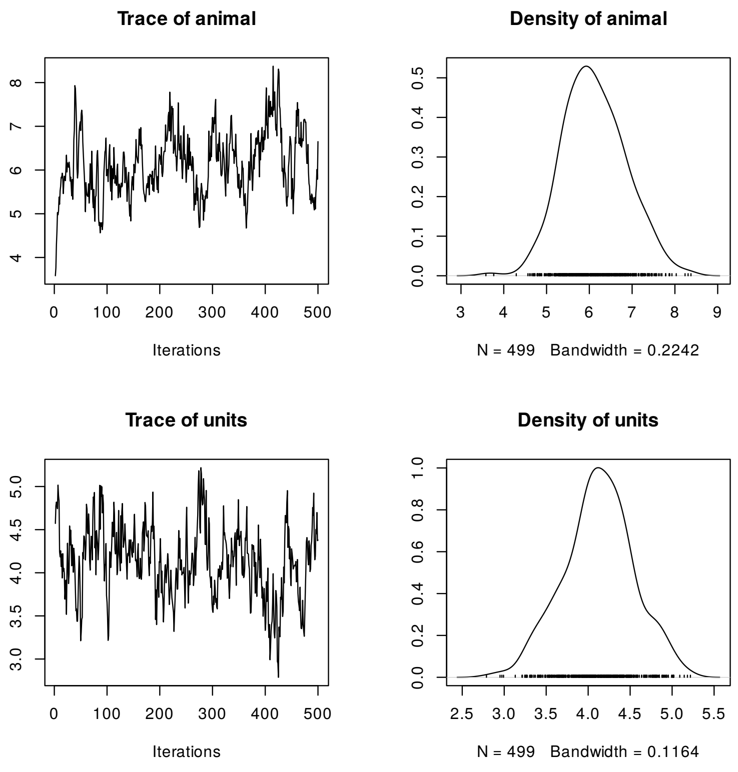

Figure 4: Trace of the variances with only 500 iterations.

```
       Stationarity start     p-value
       test         iteration
animal passed       1         0.656
units  passed       1         0.681

       Halfwidth Mean Halfwidth
       test
animal passed    5.99 0.0218
units  passed    4.18 0.0117
```

Important

This kind of convergence tests do not spare one from graphically checking the MCMC using the trace and **certainly not** from computing effective sample sizes (convergence and autocorrelation issues are mostly separate!).

Note

The "Halfwidth Mean" part of `heidel.diag` is not related to convergence, but try to assess whether the chain was run long enough to get a given precision level. It can be quite sensitive to departure from normality and is not as informative as the computation of effective sample size in the end.

3.4.3 Results from a MCMC algorithm

The output of the `MCMCglmm()` function has the following structure:

```
summary(model)
```

```
 Iterations = 10001:99991                    # MCMC parameters
 Thinning interval  = 10
 Sample size  = 9000                         # Actual sample size (not effect.)

 DIC: 4693.942                               # Somewhat like AIC

 G-structure:  ~animal                       # Random effects section

       post.mean l-95% CI u-95% CI eff.samp
animal     5.992    4.529    7.408     4485  # Here is an effective sample size

 R-structure:  ~units                        # Residual variance section

      post.mean l-95% CI u-95% CI eff.samp
units     4.183    3.399    5.014     4792

Location effects: Tarsus ~ 1                # Fixed effects section

            post.mean l-95% CI u-95% CI eff.samp  pMCMC
(Intercept)     48.71    48.20    49.24     8159 <1e-04 ***
```

The first part reminds us the characteristics of the sample. Then, the function gives the DIC (Deviance Information Criterion) associated to the model. This DIC may be used for model selection[11], and is akin to the AIC (with different properties). The rest of the output is separated into three sections: the first, `G-structure`, informs us about the variance estimates of the random effects (here, we only have V_A, called `animal`); the second, `R-structure`, is about the residual variance estimation (V_R here called `units`); and the third section, `Location effects`, gives us the results regarding the fixed effects (here only the population mean μ called `Intercept`).
Concerning the two first parts, we get various information about our estimations. The column gives us the posterior mean of the posterior distribution (actually, it is simply the mean of the MCMC sample). However, it has to be noted that the median is sometimes a better summary statistics than the mean, especially for very asymmetric posterior distribution. We then have the lower and upper limits of the 95% credible interval (i.e. our parameter has a posterior probability of 0.95 to lie within this interval). Finally, the function gives us the effective sample size associated to the parameter.
Concerning the fixed effects, we have the same information plus a `pMCMC` column. This latter provides a degree of significance for the parameter being away from zero[12]. It is not exactly a p-value, but provides the same kind of information. Here, the `pMCMC` is very low indicating strong significance away from zero (which also makes sense when looking at the credible interval).

3.5 Computation of heritability estimate

One big advantage of the MCMC is the possibility to straightforwardly compute the posterior distribution of any function of the variance components. Indeed we can do this by directly computing on the MCMC samples: because the computation is applied on each element of the MCMC samples, it is the same as if we computed the quantity during the model estimation, using each of the parameters value. In doing so, we obtain a MCMC sample of the posterior distribution of the transformed

[11]See [this discussion](https://stat.ethz.ch/pipermail/r-sig-mixed-models/2012q2/018096.html) however for the limits of DIC and how it is computed in MCMCglmm, and many other software.

[12]To be precise, it is twice the the posterior probability of the parameter being positive or negative, whichever is the smallest.

parameter (e.g. the heritability). Thus we have a simple way to obtain the posterior distribution of the heritability:

```
herit <-
    model[["VCV"]][ , "animal"] /
    (model[["VCV"]][ , "animal"] + model[["VCV"]][ , "units"])
```

We can then use all the tools we just mentioned on the heritability:

```
effectiveSize(herit)
```

```
    var1
4277.756
```

```
mean(herit)
```

```
[1] 0.5873107
```

```
HPDinterval(herit)   # Compute the 95% credible interval
```

```
         lower     upper
var1 0.4867748 0.6762358
attr(,"Probability")
[1] 0.95
```

We can also plot the trace and the posterior density using `plot` (Figure 5). In the end, the heritability of the tarsus length of our phoenix population is about 0.59 with 95% of posterior probability to lie between 0.49 and 0.68.

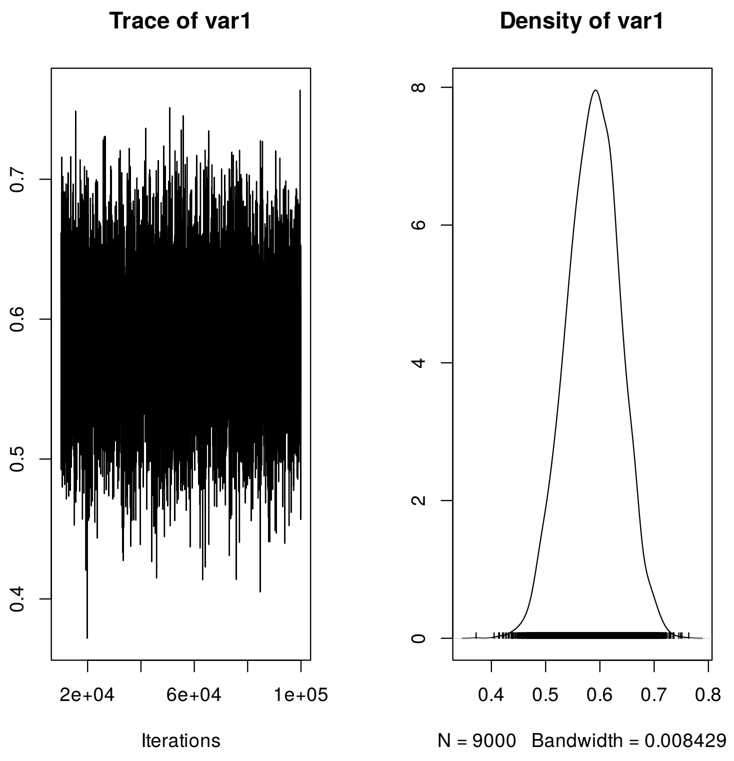

Figure 5: Trace (left) and posterior density (right) of the heritability.

3.6 Adding random effects

Now, we might want to check whether there is an effect of the habitat structure in our inference. If related individuals tend to be found clustered in space, then we might confound the influence of the habitat with the influence of their genetics. This, for example, happens when the habitat is structured and offspring tend to stick with their parents' habitat. Here, we will account for this by including another random effect in our model: the effect of the wood the individuals are found in.

```
priorRE <- list(R = list(V = 1, nu = 0.002),
                G = list(G1 = list(V = 1, nu = 0.002),
                         G2 = list(V = 1, nu = 0.002)))
modelRE <- MCMCglmm(Tarsus ~ 1,
                    random   = ~ animal + Wood,
                    family   = "gaussian",
                    prior    = priorRE,
                    pedigree = pedigree,
                    data     = data,
                    nitt     = 100000,
                    burnin   = 10000,
                    thin     = 10)
```

We still do the usual checks and calculate the heritability including all random effects contained in `VCV` (`rowSums` sums on all the different effects):

```
heritRE <- modelRE$VCV[, "animal"] / rowSums(modelRE$VCV)
```

We here see that the heritability is now estimated around 0.42, which is small than the 0.59 above:

```
mean(heritRE)
```

```
[1] 0.4167625
```

```
HPDinterval(heritRE)
```

```
         lower     upper
var1 0.3041095 0.5213566
attr(,"Probability")
[1] 0.95
```

Through this example, we can see the importance of a well defined model, including all the possibly confounding effects to correctly estimate heritability.

3.7 Adding fixed effects

Now, what if we wanted to add some fixed effects as well as random effects? This is easily done in MCMCglmm, we simply need to use the modelling formula, just as we would anywhere else in R:

```
modelFE <- MCMCglmm(Tarsus ~ Birth_Temp,
                    random   = ~ animal + Wood,
                    family   = "gaussian",
                    prior    = priorEC,
                    pedigree = pedigree,
                    data     = data,
                    nitt     = 100000,
                    burnin   = 10000,
                    thin      = 10)
summary(modelFE)
```

```
  ...              ...               ...              ...


  ...              ...               ...              ...


 Location effects: Tarsus ~ Birth_Temp

            post.mean l-95% CI u-95% CI eff.samp  pMCMC
(Intercept)   45.0754  44.1486  46.0318     9000 <1e-04 ***
Birth_Temp     0.1825   0.1554   0.2095     9000 <1e-04 ***
```

We now have additional content in the “Location effects” part of the summary, which provides the estimates for the slope of birth temperature on tarsus length.

Now, this might be problematic, because explaining such effect of temperature means that our residual variance must have decreased a bit:

```
mean(modelRE[["VCV"]][ , "units"])
mean(modelFE[["VCV"]][ , "units"])
```

```
[1] 2.678317
[1] 1.988799
```

Because of that, if we compute the denominator of the heritability (a.k.a. the total phenotypic variance V_P) as we did before, the heritability would be increased (Wilson 2008), although the additive genetic variance stayed roughly the same:

```
mean(modelRE[["VCV"]][ , "animal"])
mean(modelFE[["VCV"]][ , "animal"])
```

```
[1] 4.369585
[1] 4.306073
```

```
heritFE_naive <-
    modelFE[["VCV"]][ , "animal"] / rowSums(modelFE[["VCV"]])
mean(heritRE)
mean(heritFE_naive)
```

```
[1] 0.4167625
[1] 0.4545613
```

This might seem like not that much of an inflation here, but it could be much worse if our fixed effects explained most (or even just a substantial part) of the residual variance.

Fortunately, as explained above, there is a solution to that: we can compute the variance V_F arising from the fixed effects as the variance of the predicted values (Nakagawa & Schielzeth 2013; de Villemereuil et al. 2018). We could use the `predict()` function for that, but since this function was designed to do many different things, it is not necessarily very optimal for this simple computation. So here is a more efficient way to do it:

```
compute_varpred <- function(beta, design_matrix) {
  var(as.vector(design_matrix %*% beta))
}
```

```
X  <- modelFE[["X"]]
vf <- apply(modelFE[["Sol"]], 1, compute_varpred, design_matrix = X)
```

Now, let's see if this allows us to correct our problem:

```
vpRE <- rowSums(modelRE[["VCV"]])
vpFE <- rowSums(modelFE[["VCV"]]) + vf
mean(vpRE)
mean(vpFE)
```

```
[1] 10.5561
[1] 10.26802
```

That seems much more comparable, doesn't it? Let's see the impact on our heritability then:

```
heritFE <- modelFE[["VCV"]][ , "animal"] / vpFE
mean(heritFE)
```

```
[1] 0.4221658
```

We have fixed our inflation problem!

3.8 Using a custom covariance matrix: the dominance effect

3.8.1 *Adding a custom variance-covariance matrix, possible?*

When we specify a random effect to the function `MCMCglmm()`, it assumes a simple (identity) variance-covariance matrix between the effects. An exception is when we use `animal` and provide a pedigree, of course. But we can provide a custom variance-covariance matrix for a particular random effect, using the `ginverse` argument. It can be used to provide a genomic relatedness matrix (GRM) instead of a pedigree for example. Here, we will illustrate how to do that by fitting adding a dominance effect in the model.

3.8.2 *Computation of the dominance matrix D*

The first thing is to compute this dominance matrix. In absence of inbreeding, this matrix is made of 1 on the diagonal, 0.25 when individuals of the row and the column are full-siblings and 0 elsewhere. Note that, from the dominance perspective, parent/offspring and half-sibling relationships are null (such that only full-siblings matter)[13]. In order to compute this dominance matrix, a R package exits, called nadiv (Wolak 2012):

```
library(nadiv)
listD <- makeD(pedigree)
Dinv  <- listD[["Dinv"]]
```

```
starting to make D....done
starting to invert D....done
```

We now have the `Dinv` matrix (i.e. the inverse of the dominance matrix). The dominance matrix can be obtained using `listD$D`. A last thing that we need to do is create a column that will refer to the individuals, since this dominance matrix contains a value for each pair of individuals (just as the $\boldsymbol{A}$ matrix). We can do that very simply by copying the column containing the individual IDs, the `animal` column:

[13]Actually, double first cousins have a dominance correlation of $\frac{1}{16}$, but they are much less common compared to full-siblings in nature.

```
data[["Dom"]] <- data[["animal"]]
```

3.8.3 How to implement dominance effects in MCMCglmm?

So, now, we have everything we need. We just need to link everything together in our model call. The column containing the IDs is named `Dom` and our (inverse) dominance variance-covariance matrix is named `Dinv`. We can link the two using the `ginverse` argument. Remembering to modify the prior to add another random effect, the call will look like this:

```
priorD <- list(R = list(V = 1, nu = 0.002),
               G = list(G1 = list(V = 1, nu = 0.002),
                        G2 = list(V = 1, nu = 0.002),
                        G3 = list(V = 1, nu = 0.002)))
modelD <- MCMCglmm(Tarsus ~ Birth_Temp,
                   random    = ~ animal + Wood + Dom,
                   family    = "gaussian",
                   prior     = priorD,
                   pedigree  = pedigree,
                   data      = data,
                   ginverse  = list(Dom = Dinv),         # Note this line
                   nitt      = 100000,
                   burnin    = 10000,
                   thin      = 10)
```

Note The model will take quite a while to run...

If we look at the summary, we can that the dominance effect seems to account for almost all of the residual variance in the previous models. But we can also see a few problems with the effective size of the dominance effect, and even more, of the residual variance (actually, you might also find some issues about convergence.).

```
summary(modelD)
```

```
 Iterations = 10001:99991
 Thinning interval  = 10
 Sample size  = 9000

 DIC: -2027.532

 G-structure:  ~animal

       post.mean l-95% CI u-95% CI eff.samp
animal     4.205    3.288    5.168     3858

               ~Wood

     post.mean l-95% CI u-95% CI eff.samp
Wood      3.25    1.686    5.227     9000

               ~Dom
```

```
    post.mean l-95% CI u-95% CI eff.samp
Dom     2.129    1.557    2.751    375.9                  # Here

 R-structure:  ~units

      post.mean  l-95% CI u-95% CI eff.samp
units   0.06377 0.0001456   0.3396    72.45               # Here

 Location effects: Tarsus ~ Birth_Temp

            post.mean l-95% CI u-95% CI eff.samp  pMCMC
(Intercept)   45.0800  44.1159  46.0030     9000 <1e-04 ***
Birth_Temp     0.1820   0.1557   0.2093     9000 <1e-04 ***
```

The inference is behaving a bit erratically because *(i)* estimating the dominance effect is relatively difficult, especially in the very simple and relatively small dataset here, *(ii)* there is, in truth[14], no dominance effect to explain here and *(iii)* as a result, there is some difficulty here to distinguish the "dominance effect" from purely residual variance. Since this was just to illustrate how to include a custom variance-covariance matrix, let's just drop this failure of an inference here.

4 Generalised animal model using MCMCglmm

4.1 A Poisson model

4.1.1 The context

To illustrate how to work with a count trait using a Poisson model, we will use a trait of our phoenix dataset: the number of revival. Contrary to popular belief[15], most phoenix do not revive from their ashes and when they do, they are limited in the number of times they can. In other words, the number of revivals is limited and differ from individuals to individuals. The number of revivals follow the distribution shown in Figure 6, which is very likely to be well fitted using a Poisson distribution (besides the trait being a count trait, of course).

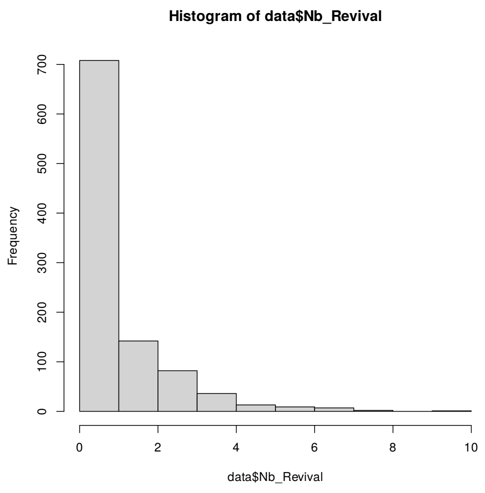

Figure 6: Distribution of the number of phoenix revivals in our dataset.

4.1.2 Choosing a prior

If you recall Equation 13, describing the log-Poisson GLMM, the latent trait is modelled on the log-scale, i.e. before taking its exponential value. This has an important consequence for our choice of a prior: because of this exponential, we expect the values of the latent trait, and even more their variance, to be *small*. So, we need (or rather, it would be best for us) to incorporate this prior knowledge into our actual prior distribution. In section 3.2.3, I described a parameter expanded prior that is more "gentle" for variances close to zero, so we will use that prior here:

[14]As shocking as it might come to you, phoenix do not exist and the dataset was simulated.
[15]And very conveniently for the purpose of this tutorial!

```
priorP <- list(R = list(V = 1, nu = 1),
               G = list(G1 = list(V = 1, nu = 1, alpha.mu = 0, alpha.V = 1000),
                        G2 = list(V = 1, nu = 1, alpha.mu = 0, alpha.V = 1000)))
```

4.1.3 Running the model

To run the model, we can use the same model as when we studied tarsus length, except we change the response variable for `Nb_Revival` (obviously) and the family for a Poisson distribution:

```
modelP <- MCMCglmm(Nb_Revival ~ 1,
                   random   = ~ animal + Wood,
                   family   = "poisson",        # Note the family here
                   prior    = priorP,
                   pedigree = pedigree,
                   data     = data,
                   nitt     = 100000,
                   burnin   = 10000,
                   thin     = 10)
```

```
MCMC iteration = 0
Acceptance ratio for liability set 1 = 0.000552

MCMC iteration = 1000
Acceptance ratio for liability set 1 = 0.433484

...            ...              ...         ...

MCMC iteration = 1000000
Acceptance ratio for liability set 1 = 0.379682
```

When using a GLMM, `MCMCglmm` shows the acceptance ratio for the latent trait (`liability set`), which should fluctuate around 0.44[16]. Now, we can, as always, look at the summary of the model:

```
summary(modelP)
```

```
 Iterations = 10001:99991
 Thinning interval  = 10
 Sample size  = 9000

 DIC: 2880.338

 G-structure:  ~animal

       post.mean l-95% CI u-95% CI eff.samp
animal    0.1632   0.0561    0.284     1309

               ~Wood

     post.mean  l-95% CI u-95% CI eff.samp
Wood   0.01603 1.343e-11  0.05111     2110
```

[16]For multi-trait models, the optimal acceptance ratio decreases, with a limit at 0.22 for large dimensions.

```
 R-structure:  ~units

      post.mean l-95% CI u-95% CI eff.samp
units     0.282    0.158    0.407    951.7

 Location effects: Nb_Revival ~ 1

            post.mean l-95% CI u-95% CI eff.samp pMCMC
(Intercept)  -0.04283 -0.18289  0.10017     3786 0.559
```

After doing all the diagnostic checks describe above, we can see that the MCMC went quite well.

4.1.4 Computing the heritability

As mentioned in section 2.5.2, we need to account for the assumptions and structure of GLMM to compute heritability. The way we computed the heritability up until now corresponds to the heritability of the latent (virtual) trait (let's call it h^2_{lat}), not the actual phenotypic trait:

```
heritP_lat <- modelP[["VCV"]][ , "animal"] / rowSums(modelP[["VCV"]])
mean(heritP_lat)
HPDinterval(heritP_lat)
```

```
[1] 0.3535495
         lower     upper
var1 0.1341322 0.5907295
attr(,"Probability")
[1] 0.95
```

This is all well, but not necessarily what we were interested in. What we want is not the heritability of some virtual, latent trait we assumed the existence of, but of our phenotype.

In order to get to the heritability of the trait (let's call it h^2_{obs}), we can use `QGglmm`. The most simple way to do that would be to compute point estimates for V_{A} and V_{P} and provide them to `QGglmm`:

```
library(QGglmm)
mu_est <- mean(modelP[["Sol"]][ , "(Intercept)"])
va_est <- mean(modelP[["VCV"]][ , "animal"])
vp_est <- mean(rowSums(modelP[["VCV"]]))
QGparams(mu    = mu_est,
         var.a = va_est,                   # IMPORTANT
         var.p = vp_est,                   # This is not the best way,
         model = "Poisson.log")            # please the next paragraph!
```

```
[1] "Using the closed forms for a Poisson - log model."
  mean.obs  var.obs var.a.obs    h2.obs
1 1.206544 2.059543 0.2375566 0.1153443
```

We can see that the heritability of the trait (`h2.obs`) is much lower than the heritability computed on the latent trait computed above (`heritP_lat`). But this approach has two important limitations. First, we should not use point estimates to compute derived quantities such as the heritability, but use the whole posterior distribution. Second (though the point is similar), we did not obtain the uncertainty around h^2_{obs}, which would be nice to have.

The good news is that we can do it properly, again using the MCMC samples to compute the posterior distribution for h^2_{obs}, as we've done throughout section 3. It's not as easy though, as because it is a bit of a complex function, we cannot directly provide a vector to `QGparams`. We can loop over it though. A good way to do that in R, it to use an `*apply` function or the library `purrr`. I find the syntax of the latter to be more consise, so we will use it here, but see the package vignette for another strategy using `mapply()`. So, we will use `pmap_dfr()`, which allows for providing a series of parameters for a function that returns a data.frame and then "stacks" the results into one big data.frame:

```r
library(purrr)
paramsP <-
    pmap_dfr(list(mu    = modelP[["Sol"]][ , "(Intercept)"],  # - This is the list
                  var.a = modelP[["VCV"]][ , "animal"],       # | which provides our
                  var.p = rowSums(modelP[["VCV"]])),          # | posterior distr.
             QGparams,                                        # - Calling the function
             model = "Poisson.log",                           # - Other arguments
             verbose = FALSE)                                 # | for QGparams
mean(paramsP[["h2.obs"]])
HPDinterval(as.mcmc(paramsP[["h2.obs"]]))
```

```
[1] 0.1146387
          lower     upper
var1 0.03897805 0.1912676
attr(,"Probability")
[1] 0.95
```

Now, we obtained exactly what we wanted! It required a bit of coding effort in R, but I hope this is relatively easy to follow here.

4.1.5 Adding the fixed effects

What if we wanted to add a fixed effect in the model? We learned before that we could compute the V_F component and add that to the denominator. For GLMM however, things are a bit more complicated and adding V_F when computing V_P is not enough. This is because of the non-linearity introduced by the link function (logarithm here)[17]. What we need to do, instead, is to provide directly the predicted values on the latent scale to QGparams, which will use them to account for the influence of the fixed effects.

But first, let's add a fixed effect in our model. This will be the lunar phase at the date of birth of the individuals:

```r
modelPFE <-
    MCMCglmm(Nb_Revival ~ Birth_Moon,
             random   = ~ animal + Wood,
             family   = "poisson",
             prior    = priorP,
             pedigree = pedigree,
             data     = data,
             nitt     = 100000,
             burnin   = 10000,
             thin     = 10)
```

[17]For more information about this, please see de Villemereuil et al. (2016) and de Villemereuil et al. (2018).

Now, we can compute the predicted values for each iteration of the MCMC (using `map()` from the `purrr` library[18]):

```r
X <- modelPFE[["X"]]
predict <- map(1:nrow(modelPFE[["Sol"]]),
               ~ as.vector(X %*% modelPFE[["Sol"]][., ]))
```

Then, it becomes relatively easy to use these predicted values in `QGparams()` instead of providing the intercept `mu` as above:

```r
paramsPFE <-
    pmap_dfr(list(predict = predict,
                  var.a   = modelPFE[["VCV"]][ , "animal"],
                  var.p   = rowSums(modelPFE[["VCV"]])),
             QGparams,
             model = "Poisson.log",
             verbose = FALSE)
mean(paramsPFE[["h2.obs"]])
HPDinterval(as.mcmc(paramsPFE[["h2.obs"]]))
```

```
[1] 0.1005345
          lower     upper
var1 0.04361177 0.1581883
attr(,"Probability")
[1] 0.95
```

And done! We properly integrated the influence of the fixed effects, all while using the posterior distribution from the MCMC inference. Pretty neat, isn't it?

4.2 A binomial model

Phoenix come into two colours: golden or white plumage. Is this heritable? Let's study this.

4.2.1 Some theory

The usual model to study binary traits is something called a *threshold model*. In this model, we assume a (virtual, again) Gaussian trait named "liability", which when it goes (or not) over a given threshold triggers the expression of one of the binary phenotype we want to study (Figure 7, left). This model was developed by Wright (1934). An interesting thing is that when we use a GLMM with a binomial distribution and a probit link, we use a model that is equivalent to this threshold model (Figure 7, right; see de Villemereuil et al. 2016 and de Villemereuil 2018 for more explanation on this subject). The only thing we need to go from the latent scale of a GLMM to the liability scale of a threshold model is to add the so-called "link variance" to the denominator (which is equal to 1 for a probit link, and $\frac{\pi^2}{3}$ for a logit link, see e.g. Nakagawa & Schielzeth 2010). There are two main interests in computing the heritability on the liability scale, rather than on the latent scale. First, given the historical importance of the threshold model, it makes the heritability estimates of binary traits available in the literature more comparable. Second, since the observed trait is assumed to be a deterministic output of the liability scale (see Figure 7), it can be considered as "closer" to the actual phenotype.

[18] `lapply()` works just as well, of course!

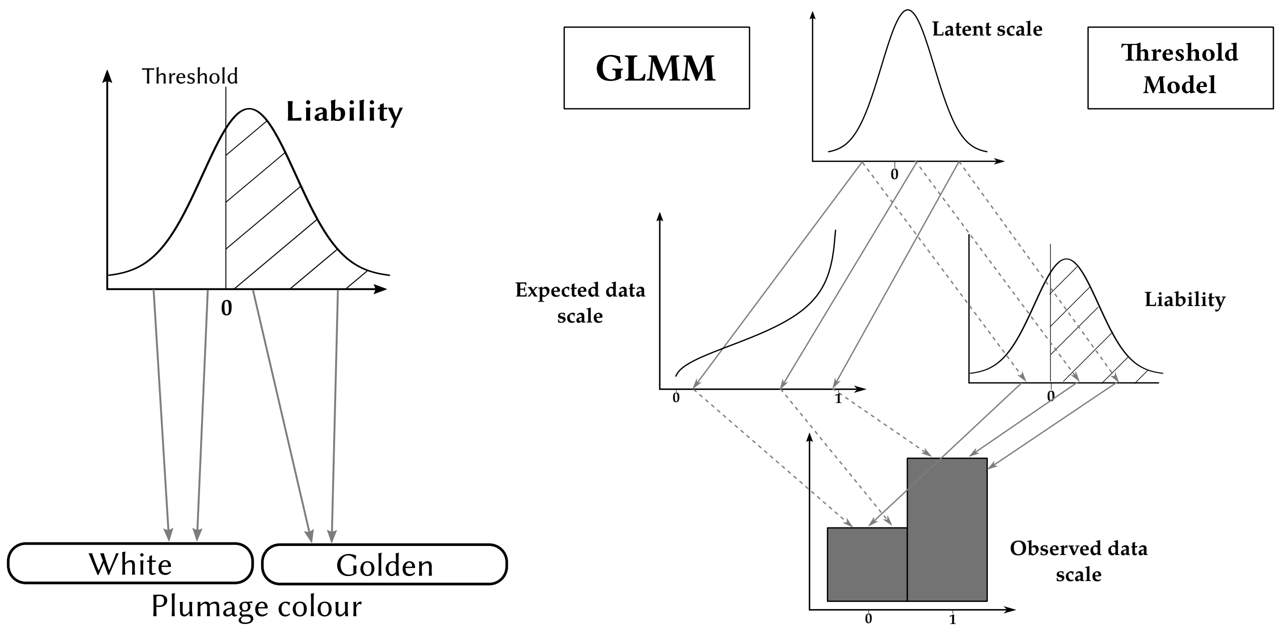

Figure 7: Left, threshold model: a trait (named "liability") is assumed to be Gaussian and a threshold determines whether the observed phenotype is of one category (white or 0) or the other (golden or 1). **Right, equivalence with GLMM:** There is an equivalence between a GLMM with a probit link and the threshold model, but the mechanism is slightly different. The latent scale of a GLMM and the liability of a threshold model are not strictly the same. While the GLMM deterministically transforms the latent scale into probability (plain grey arrows on the left) then probabilistically into the observed data scale (dashed grey arrows on the left), the threshold model equivalent first makes a "noiser" version of the latent scale (obtaining the liability scale, dashed grey arrow on the right) and then deterministically define the observed phenotype (plain grey arrow). This is a very complicated graph to say that the latent scale of a GLMM is not exactly the liability of the threshold model, we need to add a bit more variance to the latent scale to get the liability scale, see the main text.

Fortunately, `MCMCglmm` has a "threshold" family (Hadfield 2015) that reduces the latent and liability scale to the same thing, so if we use that, we do not have to bother much about all these differences[19].

4.2.2 Choosing a prior

Binomial models require somewhat specific priors. Indeed, it is impossible to evaluate the residual variance V_R for these kind of models[20]. Because of this identifiability issue, we usually fix V_R to 1 and estimate the variance of other random factors. Because of this, it becomes tricky to define a good prior for V_A (and other possible random effects variance). On Figure 8, we see the difference between the inverse-Gamma prior we used for the main part of the tutorial and a new prior (a χ^2 with 1 degree of freedom). We can see that while the first is heavily unbalanced and putting almost all of its weight on 1, the second is more equally spread along the $[0, 1]$ interval. I actually shown a while ago that this second prior resulted in a better inference for binary traits (de Villemereuil et al. 2013, Appendix B). I thus advice to use this χ^2 prior distribution when estimating heritability of binary data. As always though, the influence of the prior distribution fade away with a sufficient sample size. To use this new prior, we use the parameter extension with the supplementary parameters `alpha.mu` and `alpha.V`:

[19]Why bothering in explaining all this, then, you wonder? Well, first, I believe all this is important to understand this idea of "link variance" that is mentioned in the literature, and which only has a sense in this context of a threshold/GLMM equivalence. Second, you might want to compare `MCMCglmm` results to another program, then you should recall to either use another family like "ordinal" or remember to add this link variance (+1) to your denominator in that other program.

[20]Indeed, whatever the value of $V_\mathrm{P} = V_\mathrm{A} + V_R$, the variance of the binary data will always be $p(1-p)$ where p is the probability of success (and does not depend on V_P).

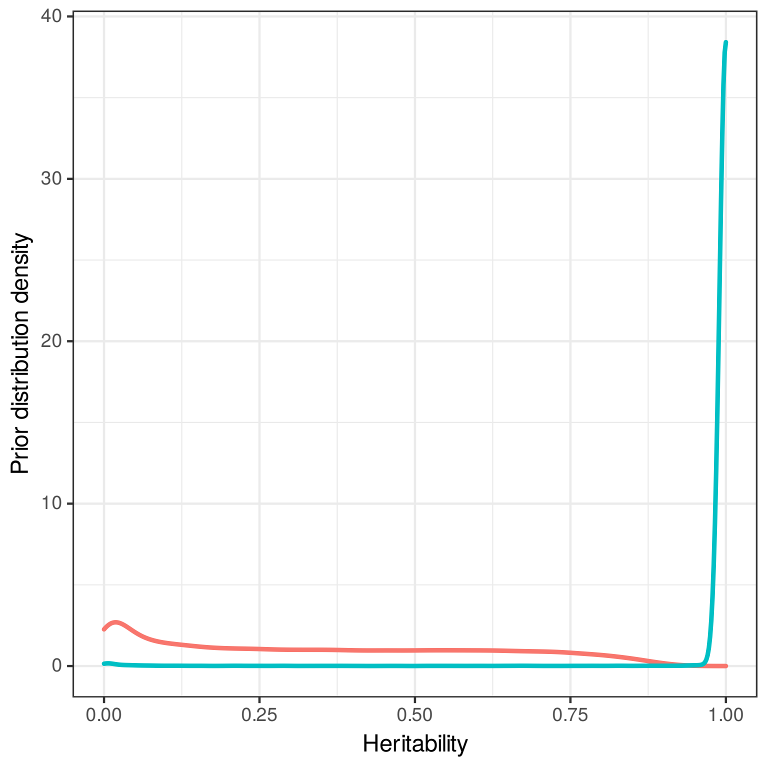

Figure 8: Prior density for the heritability estimate when the residual variance V_R is fixed to 1 and the other random component (e.g. V_A) follows either an inverse-gamma (0.001, 0.001) distribution (in blue) or a chi-square distribution with 1 degree of freedom (in red). We can see that the first is highly concentrated around 1, while the latter is more equally spread over the $[0, 1]$ continuum.

```
prior <- list(R = list(V = 1, fix = 1),
              G = list(G1 = list(V = 1, nu = 1000, alpha.mu = 0, alpha.V = 1)))
```

4.2.3 Running the model

To run the model, we thus need to define this new prior and use the correct response variable and family. Regarding the family, we will use “threshold” here.

```
priorB <- list(R = list(V = 1, fix = 1),
               G = list(G1 = list(V = 1, nu = 1000, alpha.mu = 0, alpha.V = 1),
                        G2 = list(V = 1, nu = 1000, alpha.mu = 0, alpha.V = 1)))
modelB <- MCMCglmm(White ~ 1,
                   random   = ~ animal + Wood,
                   family   = "threshold",
                   prior    = priorB,
                   pedigree = pedigree,
                   data     = data,
                   nitt     = 100000,
                   burnin   = 10000,
                   thin     = 10)
```

Note

Apart from “threshold”, other possible families include “categorical” which uses a logit link and “ordinal”, which also uses a probit link, but doesn’t merge the latent and liability scales as explained above. Note that “threshold” usually results in better mixing of the MCMC (Hadfield 2015).

We can now look at the summary of the model:

```
summary(modelB)
```

```
 Iterations = 10001:99991
 Thinning interval  = 10
 Sample size  = 9000

 DIC: 1056.902

 G-structure:  ~animal

      post.mean l-95% CI u-95% CI eff.samp
animal     1.147   0.4088    2.044    707.9

               ~Wood

     post.mean l-95% CI u-95% CI eff.samp
Wood    0.3052  0.09272   0.5745     1677

 R-structure:  ~units

      post.mean l-95% CI u-95% CI eff.samp
units         1        1        1        0      # Note here

 Location effects: White ~ 1

            post.mean l-95% CI u-95% CI eff.samp  pMCMC
(Intercept)   -0.7809  -1.1618  -0.4375     3481 <1e-04 ***
```

You might notice that the effective sample size for `units` is 0. This is because we fixed the residual to 1, of course, so nothing to worry about.

4.2.4 Computing the heritability

We can now compute the heritability of our trait on the liability scale. Since we used the "threshold" model, we can actually compute it the same way we always did:

```
heritB_liab <-
    modelB[["VCV"]][ , "animal"] / rowSums(modelB[["VCV"]])
mean(heritB_liab)
HPDinterval(heritB_liab)
```

```
[1] 0.4524734
         lower     upper
var1 0.2880247 0.6171448
attr(,"Probability")
[1] 0.95
```

IMPORTANT

It works here only because we used the "threshold" family! If we used the "ordinal" family, the denominator would be:
`rowSums(modelB[["VCV"]]) + 1`
If we used the "categorical" family, the denominator would be:
`rowSums(modelB[["VCV"]]) + (pi^2)/3`

We can also want to compute the heritability on the observed data scale, i.e. for the actual binary trait. To do so, as above, we can use the QGglmm package, integrating over the posterior distribution as above:

```
paramsB <-
    pmap_dfr(list(mu    = modelB[["Sol"]][ , "(Intercept)"],
                  var.a = modelB[["VCV"]][ , "animal"],
                  var.p = rowSums(modelB[["VCV"]]) - 1),     # Note the - 1 here
             QGparams,
             model = "binom1.probit",
             verbose = FALSE)
mean(paramsB[["h2.obs"]])
HPDinterval(as.mcmc(paramsB[["h2.obs"]]))
```

```
[1] 0.2617547
         lower     upper
var1 0.1622869 0.3584228
attr(,"Probability")
[1] 0.95
```

Note

Notice the `- 1` in the definition of `var.p`. This is because (once again) we used the "threshold" family in `MCMCglmm`. But also notice we used "binom1.probit" as a description of the model. In order to resolve this discrepancy, it is needed to remove 1 from `var.p` to account for the "link variance" that `QGparams` will add in its transformation.

4.2.5 *Which estimate to report?*

A question that often arise is which estimate to report? Contrary to the log-Poisson model above, where the heritability on the latent scale is almost always unsatisfying, the case here is more subtle.

As I explained above, the liability scale has a deterministic relationship with the binary trait, in the sense knowing its value and the threshold is enough to perfectly predict the trait[21]. Imagine that your binary trait is not *naturally* binary (e.g. survival or dispersal), but was made binary when it was measured (e.g. small/large). Then it could be argued that the actual trait *is* the liability and reporting heritability on this scale would make sense. Another reason to report the heritability on the liability scale is that it does not depend on the prevalence of the values (0/1) of the binary trait. A last reason is tradition: almost all heritability estimates for binary traits in the literature have been reported on the liability scale, so for the sake of comparison, there is a strong value in carrying on the tradition.

At the same time, completely ignoring the heritability on the observed data scale can result in spurious conclusions. Imagine you are a breeder trying to get rid of a disease of strong genetic origin in your stock. Let's say it affects roughly 1% of the individuals and the heritability of such disease on the liability scale is around 0.8. Given such a high estimate, it should be relatively easy to select against the disease, right? Well, removing the all afflicted individuals from the breeding population (strongest possible selection) would only shift the prevalence from 1% to 0.94% in the next generation. Each generation, it will become more and more difficult to reduce the prevalence. This

[21]Of course, since the liability is a virtual, non-existing scale, this determinism is also very virtual...

is because selection is limited by the low prevalence of the disease[22], and computing the heritability on the observed data scale would have revealed it right away, since it equals 0.058 in this scenario!

In the end, which estimate is best to report and analyse depends on the context. At the very least, providing the average value (prevalence of the trait in the population) is required, because it is the key to navigate from a scale to the other (see Dempster & Lerner 1950).

5 Multi-traits models in MCMCglmm

5.1 A Gaussian multi-traits model

In our phoenix dataset, we have two Gaussian traits: the tarsus length we have studied above and the weight of the individuals. If tarsus length is a good predictor of overall individual size, then we would expected weight and tarsus length to be positively correlated[23]. We could even expect such correlation to be positive both at the genetic and residual levels. Let's try to perform the analysis.

5.1.1 A word on priors

As always, before anything, we need to look into priors. Here, we are considering simple variances, but variance-covariance matrices (see Section 2.6). In two dimensions, such a variance-covariance matrix is shaped like:

$$\begin{pmatrix} V_1 & C_{1,2} \\ C_{1,2} & V_2 \end{pmatrix} \tag{21}$$

where V_i is the variance (for example additive genetic) associated to the trait i and $C_{1,2}$ is the covariance (for example genetic) between trait 1 and trait 2. We thus need a kind of prior distribution relevant for matrices. To do so, `MCMCglmm` use a matrix distribution called *inverse Wishart*. It is difficult to keep using our very non-informative prior, especially as multi-trait models are complex and thus require a bit more help to work properly.

Note

If you'd really like to use a prior similar to the we've been mostly using all along, it would look like this:

```
list(R = list(V = diag(2) * 0.002 / 1.002, nu = 1.002),
     G = list(G1 = list(V = diag(2) * 0.002 / 1.002, nu = 1.002)))
```

There might be a few bumps in the MCMC with this one, but it is less informative than the one we will use below.

So, I often use a more "gentle" prior for multi-trait models:

```
prior <- list(R = list(V = diag(2), nu = 2),
              G = list(G1 = list(V = diag(2), nu = 2)))
```

The idea here is that when `nu` is at least equal to the number of dimensions, then the inverse-Wishart is behaving more "nicely".

[22]The reason this is so difficult is that the alleles at the numerous genes involved in the quantitative trait segregate at each generation, always generating new combinations that exceeds the threshold, while at the same time making a lot more combinations that are close but below the threshold (and ready to "strike" at the next generation).

[23]Basically, a very pompous way of saying big individuals weight more…

NOTE

This prior is more informative than the other one above, notably for small variances. If you have traits with very large variances, or widely different variances, scaling them to a variance of 1 might be a good option (then rescaling the estimates is only a matter of multiplying them by the original trait variances).

5.1.2 Using `MCMCglmm()` with a multi-traits model

Let's first look at the variances of our traits :

```
var(data[["Tarsus"]])
var(data[["Weight"]])
```

```
[1] 10.3669
[1] 290.3217
```

As we can see, the variances are very different for the two traits. Now, `MCMCglmm()` can accommodate that quite nicely, but if we want to use the more informative "gentle" prior above, a variance of 290 of one the trait is a bit high and rescaling the traits to the same variance unit might help the MCMC process as well as help us define our prior. Let's try:

```
data[["Tarsus_scl"]] <- scale(data[["Tarsus"]])
data[["Weight_scl"]] <- scale(data[["Weight"]])
```

To provide multiple traits to `MCMCglmm()`, we will use the `cbind()`[24] command in the response part of the formula:

```
priorM <- list(R = list(V = diag(2), nu = 2),
               G = list(G1 = list(V = diag(2), nu = 2)))
modelM <- MCMCglmm(cbind(Tarsus_scl, Weight_scl) ~ trait - 1,
                   random   = ~ us(trait):animal,
                   rcov     = ~ us(trait):units,
                   family   = c("gaussian", "gaussian"),
                   prior    = priorM,
                   pedigree = pedigree,
                   data     = data,
                   nitt     = 100000,
                   burnin   = 10000,
                   thin     = 10)
```

We can see how we used `cbind()` to provide two traits to the model. On the other side of the formula (after the ~), the notation `trait - 1` states that we want to model a different intercept for each trait. Note that the argument `family` now requires a vector with the data distribution of each trait. Finally, the argument `random` has changed and a new argument `rcov` appeared: these two arguments define the structure of the variance-covariance matrix for the random effects (`random`) or the residual variances (`rcov`). The command `us(trait):animal` states that we define these matrix exactly as written is Equation 21. This is the role of the function `us()` (for unstructured variance model). If we simply had used `random=~animal`, we would have defined the following (somewhat

[24]Careful readers will notice that in section 2.6, we mentioned that the traits were "stacked" while `cbind()` doesn't do that, since it is supposed to collate the vectors into different columns of a data frame. This is true, the R syntax here does not translate well what is happening mathematically. More information is available in the MCMCglmm Course Note(Hadfield 2016)

silly) variance-covariance matrix:

$$\begin{pmatrix} V & V \\ V & V \end{pmatrix} \tag{22}$$

Doing so, we assume that both traits have the same variance ($V_1 = V_2 = V$) and are perfectly correlated ($\rho_{1,2} = 1$). Beside the function `us()`, the function `idh()` (for heterogeneous identity variance model) exists, which fixes the covariances to 0. Indeed the call `random=~idh(trait):animal` defines the following matrix:

$$\begin{pmatrix} V_1 & 0 \\ 0 & V_2 \end{pmatrix} \tag{23}$$

Note

More variance models are available and listed in `?MCMCglmm` manual. It is possible to define even more complicated structures such as multiple membership model, which is outside the scope of this tutorial.

5.1.3 Results of a multi-trait model

The previous MCMC run give us the following results:

```
summary(modelM)
```

```
 Iterations = 10001:99991
 Thinning interval  = 10
 Sample size  = 9000

 DIC: 4395.43

 G-structure:  ~us(trait):animal

                                       post.mean l-95% CI u-95% CI
traitTarsus_scl:traitTarsus_scl.animal    0.5803   0.4466   0.7196
traitWeight_scl:traitTarsus_scl.animal    0.3972   0.2859   0.5106
traitTarsus_scl:traitWeight_scl.animal    0.3972   0.2859   0.5106
traitWeight_scl:traitWeight_scl.animal    0.4325   0.3146   0.5532
                                       eff.samp
traitTarsus_scl:traitTarsus_scl.animal     4702
traitWeight_scl:traitTarsus_scl.animal     4175
traitTarsus_scl:traitWeight_scl.animal     4175
traitWeight_scl:traitWeight_scl.animal     3847

 R-structure:  ~us(trait):units

                                      post.mean l-95% CI u-95% CI
traitTarsus_scl:traitTarsus_scl.units    0.4063   0.3319   0.4837
traitWeight_scl:traitTarsus_scl.units    0.2945   0.2256   0.3619
traitTarsus_scl:traitWeight_scl.units    0.2945   0.2256   0.3619
traitWeight_scl:traitWeight_scl.units    0.5753   0.4929   0.6617
                                      eff.samp
traitTarsus_scl:traitTarsus_scl.units     5376
traitWeight_scl:traitTarsus_scl.units     5189
traitTarsus_scl:traitWeight_scl.units     5189
```

```
traitWeight_scl:traitWeight_scl.units     5450

 Location effects: cbind(Tarsus_scl, Weight_scl) ~ trait - 1

                  post.mean l-95% CI u-95% CI eff.samp   pMCMC
traitTarsus_scl  -0.24638 -0.40717 -0.08423     9000 0.00222 **
traitWeight_scl  -0.17551 -0.32480 -0.01747     9000 0.02133 *
```

In this summary, the notation `traitTarsus_scl:traitTarsus_scl` stands for the variance for (scaled) tarsus length and `traitTarsus_scl:traitWeight_scl` stands for the covariance between (scaled) tarsus length and weight. Since we scaled the traits, the variances and covariances are below 1 and directly comparable between traits. For example, we can see that the additive genetic variance of (scaled) tarsus length (0.58) is larger than the additive genetic variance of (scaled) weight (0.43), while it's the reverse regarding the residual variance (0.41 v. 0.58).

5.1.4 *Computing the quantitative genetic parameters*

A first thing that we can do is compute the unscaled variance estimates. It is possible to do by simply multiplying the variance-covariance estimates by the scaling factor, i.e. the standard deviation of the trait:

```r
# Creating another VCV to contain unscaled variance-covariance
VCV_uscl <- modelM[["VCV"]]
# Obtaining the couple of traits for each column using regular expression
# str_match() is a very useful function from the stringr package!
library(stringr)
traits <-
    str_match(colnames(VCV_uscl),
              "trait([:alpha:]+)_scl\\:trait([:alpha:]+)_scl.*")[ , c(2, 3)]
# Using map2() from the purrr package (see above) to use the trait information to
# scale the variance using the correct combination of standard deviations
# (.x and .y respectively refer to the first and second arguments of map2 here)
scl_factors <-
    map2_dbl(traits[ , 1], traits[ , 2],
             ~ attr(data[[paste0(.x, "_scl")]], "scaled:scale") *
               attr(data[[paste0(.y, "_scl")]], "scaled:scale"))
# Now, we create a matrix to host our scaling factors
scl_factors <- matrix(scl_factors,
                      nrow = nrow(VCV_uscl),
                      ncol = ncol(VCV_uscl),
                      byrow = TRUE)
# All we need to do now is apply pairwise multiplication of
# the unscaled VCV and the scaling factors
VCV_uscl <- VCV_uscl * scl_factors
colMeans(VCV_uscl)
```

```
traitTarsus_scl:traitTarsus_scl.animal traitWeight_scl:traitTarsus_scl.animal
                              6.036323                              21.918178
traitTarsus_scl:traitWeight_scl.animal traitWeight_scl:traitWeight_scl.animal
                             21.918178                             126.322487
 traitTarsus_scl:traitTarsus_scl.units  traitWeight_scl:traitTarsus_scl.units
                              4.210267                              16.132002
```

```
traitTarsus_scl:traitWeight_scl.units  traitWeight_scl:traitWeight_scl.units
                            16.132002                             166.760984
```

If we want, we can use `VCV_uscl` in the exact same way we used the VCV part of `MCMCglmm()` output:

```
effectiveSize(VCV_uscl)
heidel.diag(VCV_uscl)
```

We can also compute the heritability directly from this unscaled matrix, although using directly the VCV part of the `MCMCglmm()` output would yield the same result (as the numerator and denominator are multiplied by the same scaling factor):

```
herit_tarsus <-
    VCV_uscl[ , "traitTarsus_scl:traitTarsus_scl.animal"] /
    (VCV_uscl[ , "traitTarsus_scl:traitTarsus_scl.animal"] +
     VCV_uscl[ , "traitTarsus_scl:traitTarsus_scl.units"])
herit_weight <-
    VCV_uscl[ , "traitWeight_scl:traitWeight_scl.animal"] /
    (VCV_uscl[ , "traitWeight_scl:traitWeight_scl.animal"] +
     VCV_uscl[ , "traitWeight_scl:traitWeight_scl.units"])
mean(herit_tarsus)
HPDinterval(herit_tarsus)
mean(herit_weight)
HPDinterval(herit_weight)
```

```
[1] 0.5876398
         lower     upper
var1 0.4950912 0.6804356
attr(,"Probability")
[1] 0.95
[1] 0.429853
       lower     upper
var1 0.33518 0.5232569
attr(,"Probability")
[1] 0.95
```

As expected when we compared the scaled variance coefficients, the heritability of tarsus length is higher than the heritability of body weight. Moreover, the heritability for tarsus length is comparable to our very first model on this trait[25].

The genetic correlation can be computed just as easily (again choosing `VCV_uscl` or the scaled VCV part of the `MCMCglmm()` output doesn't matter, because the scaling factors are identical in numerator and denominator):

```
genetic_corr <-
    VCV_uscl[ , "traitTarsus_scl:traitWeight_scl.animal"] /
    sqrt(VCV_uscl[ , "traitTarsus_scl:traitTarsus_scl.animal"] *
         VCV_uscl[ , "traitWeight_scl:traitWeight_scl.animal"])
mean(genetic_corr)
HPDinterval(genetic_corr)
```

[25]Note that we have found this estimate to be inflated due to the influence of the "Wood of birth" effect, which I have not included in the model here for the sake of simplicity. Of course, a proper estimation would require such an effect to be added in the model!

```
[1] 0.7931648
         lower     upper
var1 0.7178254 0.8622758
attr(,"Probability")
[1] 0.95
```

As would be expected, there is a strong genetic correlation, around 0.79, between tarsus length (a proxy for body size) and body weight.

5.1.5 Adding fixed effects

```
modelMF <- MCMCglmm(cbind(Tarsus_scl, Weight_scl) ~ trait - 1 +
                      trait:Birth_Temp + at.level(trait, 2):Birth_Moon,
                    random   = ~ us(trait):animal,
                    rcov     = ~ us(trait):units,
                    family   = c("gaussian", "gaussian"),
                    prior    = priorM,
                    pedigree = pedigree,
                    data     = data,
                    nitt     = 100000,
                    burnin   = 10000,
                    thin     = 10)
```

You might get the following warning when running the model, which is due to the removing of one level in `at.level(trait, 2):Birth_Moon`:

```
some fixed effects are not estimable and have been removed. Use singular.ok=TRUE
to sample these effects, but use an informative prior!
```

Since this is now a multi-trait model, the equivalent of the fixed-effect variance is now a variance-covariance matrix. Let's name it F. It can be computed as follows:

```
compute_vcvpred <- function(beta, design_matrix, ntraits) {
    list(cov(matrix(design_matrix %*% beta, ncol = ntraits)))
}
X  <- modelMF[["X"]]
F <-
    flatten(
        apply(modelMF[["Sol"]], 1, compute_vcvpred, design_matrix = X, ntraits = 2)
    )
```

5.2 A non-Gaussian multi-traits model

We analysed the number of revival and plumage colour above, but do these two traits share some of their genetic bases? Maybe white phoenix tend to, genetically, revive more?

5.2.1 The prior, yet again

Remember that plumage colour is a binary trait (either gold or white), so we need to fix the residual variance to 1, using the `fix` parameter. Now, something I have omitted above is why we used `fix = 1`: the number after `fix` corresponds to the trait after which all variances will be fixed. When we have one trait, we do not have a choice (it is necessarily 1). Now that we are considering two traits, we need to change this to `fix = 2`, in order to fix the variance on the second trait (and put the binary trait in second obviously).

Note

Since we affect all the traits *after* the number provided to `fix`, all binary traits need to be placed at the end of the `cbind()` call. For example, if we have our two traits and kept `fix = 1`, then we would fix the variance for trait 1 and the trait after (trait 2), thus for all traits. In other words, we would be fixing the residual variance for the "number of revivals" Poisson trait as well (which we don't want, right?).

Another aspect is that we want to construct a prior that would be weakly informative for the small variances expected for both the Poisson (because of the exponential) and binary trait (because the residual variance is fixed to 1) on the latent scale. There are many possibilities, but one of the best way is to use, once again, the parameter expansion, while keeping the parameters weakly informative toward small variances.

So, if we account for everything we said, a possible prior would be:

```
priorM2 <-
    list(R = list(V = diag(2), nu = 2, fix = 2),
         G = list(G1 = list(V = diag(2),
                            nu = 2,
                            alpha.mu = c(0,0),
                            alpha.V = diag(2))))
```

5.2.2 Running the model

The model can be run as the other multi-trait model above, except we now also specify the non-Gaussian families we want to use:

```
modelM2 <- MCMCglmm(cbind(Nb_Revival, White) ~ trait - 1,
                   random   = ~ us(trait):animal,
                   rcov     = ~ us(trait):units,
                   family   = c("poisson", "threshold"),
                   prior    = priorM2,
                   pedigree = pedigree,
                   data     = data,
                   nitt     = 100000,
                   burnin   = 10000,
                   thin     = 10)
```

All of this should be fairly self-explaining at this point. Note that, accordingly to what I mentioned right above, we put the binary trait (`White` with family `"threshold"`) at the end. This is important, because the prior only fix the variance of the *last* trait here. We can look at the summary of the model (in practice, do not forget to do all the convergence and effective sample size diagnostics):

```
summary(modelM2)
```

```
 Iterations = 10001:99991
 Thinning interval  = 10
 Sample size  = 9000

 DIC: 3952.293
 G-structure:  ~us(trait):animal

                                       post.mean l-95% CI u-95% CI eff.samp
traitNb_Revival:traitNb_Revival.animal  0.158623  0.05773   0.2804   1276.7
traitWhite:traitNb_Revival.animal       0.007143 -0.15911   0.1845    825.1
traitNb_Revival:traitWhite.animal       0.007143 -0.15911   0.1845    825.1
traitWhite:traitWhite.animal            1.208296  0.50532   2.1279   1050.2

 R-structure:  ~us(trait):units

                                      post.mean l-95% CI u-95% CI eff.samp
traitNb_Revival:traitNb_Revival.units    0.3025  0.18007   0.4275    839.6
traitWhite:traitNb_Revival.units         0.1217 -0.03041   0.2732   1152.6
traitNb_Revival:traitWhite.units         0.1217 -0.03041   0.2732   1152.6
traitWhite:traitWhite.units              1.0000  1.00000   1.0000      0.0

 Location effects: cbind(Nb_Revival, White) ~ trait - 1

                post.mean l-95% CI u-95% CI eff.samp  pMCMC
traitNb_Revival  -0.04484 -0.17495  0.08993     2773  0.502
traitWhite       -0.77377 -1.08357 -0.46794     2917 <1e-04 ***
```

At least from the effective sample size perspective, things seem to have gone pretty well (this does not mean convergence happened though, it requires plotting the traces and/or performing some test, see our very first run of `MCMCglmm()`). The fixed residual variance is correctly assigned to the `White` component.

Obtaining the parameters on the observed data scale requires a new function from the QGglmm package, named `QGmvparams()` ("mv" for multivariate). In order to run the function over the MCMC samples, we need to format a bit the output from *MCMCglmm()*, notably transform the `"VCV"` component into real matrices:

```r
library(QGglmm)
library(purrr)

# Formatting the output into lists
# flatten() is a purrr function that removes a level of nesting in a list
# e.g. it transforms list[[1]][[1]] into just list[[1]] which contains a vector
mu <- flatten(apply(modelM2[["Sol"]], 1, list))
# Now, we can do the same with "VCV",
# but we need to format it into a matrix as well.
# Note grep("animal", ...) which collects only columns
# related to the "animal" effect.
G  <-
    flatten(
        apply(modelM2[["VCV"]][ , grep("animal", colnames(modelM2[["VCV"]]))],
          1,
          function(row) {
              list(matrix(row, ncol = 2))
          })
    )
# Now, we apply the same logic to "units"
```

```r
R <-
    flatten(
        apply(modelM2[["VCV"]][ , grep("units", colnames(modelM2[["VCV"]]))],
              1,
              function(row) {
                  row[4] <- row[4] - 1
                  list(matrix(row, ncol = 2))
              })
    )
# To obtain P we need to add G and R for each elements of their lists, easy:
P <- map2(G, R, `+`)
```

OK, this was a bit convoluted, we need now have a list for each parameter we want to provide to `QGmvparams()`, which makes it now very simple to call once for each MCMC sample:

```r
paramsM2 <-
    pmap(list(mu    = mu,
              vcv.G = G,
              vcv.P = P),
         QGmvparams,
         models = c("Poisson.log", "binom1.probit"),
         verbose = FALSE)
```

Note that we now provide the variance-covariance matrices (instead of variances) as parameters, compared to our previous call to the univariate `QGparams()`. We also need to provide a description of each of our families to `models`.

Note

The call above can take a while. But it can be easily be parallelised using the furrr package and calling `future_pmap()` instead of `pmap()`.

We now have a list that contains our parameters for each MCMC sample. However, it is better to have a separate list for each parameter instead. Fortunately, the purrr package offers a function to "transpose" a list, so we can format everything as wanted:

```r
# Transposing our list so that the first level are the parameters
paramsM2 <- transpose(paramsM2)
# And to format the output in a more simple way
# abind() from the abind package transforms a list of matrices into a 3D-array
paramsM2[["mean.obs"]] <- unlist(paramsM2[["mean.obs"]])
paramsM2[["vcv.G.obs"]] <- abind::abind(paramsM2[["vcv.G.obs"]], along = 3)
paramsM2[["vcv.P.obs"]] <- abind::abind(paramsM2[["vcv.P.obs"]], along = 3)
```

Finally! Now, we can do whatever we want with our estimates:

```r
apply(paramsM2[["vcv.G.obs"]], c(1, 2), mean)
apply(paramsM2[["vcv.G.obs"]], c(1, 2), function (vec) { HPDinterval(as.mcmc(vec))[1] })
apply(paramsM2[["vcv.G.obs"]], c(1, 2), function (vec) { HPDinterval(as.mcmc(vec))[2] })
```

```
            [,1]        [,2]
[1,] 0.232959621 0.001725556
[2,] 0.001725556 0.063911321
            [,1]        [,2]
[1,]  0.06411663 -0.04556620
```

```
[2,] -0.04556620  0.04195376
           [,1]       [,2]
[1,] 0.41544805 0.04917209
[2,] 0.04917209 0.08803096
```

Note that the covariances are not significantly different from zero, since the 95% overlaps zero (-0.045 to 0.049). If we want to focus on the heritability estimates, we can compute them as:

```
heritM2_Pois <- (paramsM2[["vcv.G.obs"]] / paramsM2[["vcv.P.obs"]])[1, 1, ]
heritM2_Bin  <- (paramsM2[["vcv.G.obs"]] / paramsM2[["vcv.P.obs"]])[2, 2, ]
mean(heritM2_Pois)
mean(heritM2_Bin)
```

```
[1] 0.1112572
[1] 0.3045554
```

Note that the estimates are comparable with those from the univariate models in subsection 4.1 and subsection 4.2, as expected, especially as the correlation between the two traits is basically zero.

5.2.3 Adding some fixed effects

Imagine that we want to account for the effect of the phase of the moon, again, on our phenotype. We can easily add the effect in our model:

```
modelM2F <- MCMCglmm(cbind(Nb_Revival, White) ~ trait - 1 + trait:Birth_Moon,
                     random   = ~ us(trait):animal,
                     rcov     = ~ us(trait):units,
                     family   = c("poisson", "threshold"),
                     prior    = priorM2,
                     pedigree = pedigree,
                     data     = data,
                     nitt     = 100000,
                     burnin   = 10000,
                     thin     = 10)
```

Now, we need to account for these fixed effects in our back-transformation of the estimates, as well as everything else we did above. The first thing we need is the posterior distribution of the predicted values:

```
X <- modelM2F[["X"]]
predict <- map(1:nrow(modelM2F[["Sol"]]),
               ~ matrix(X %*% modelM2F[["Sol"]][., ], ncol = 2))
```

Then, we can use the following steps as above, but we "merely" have to switch from using the argument `mu` to using the argument `predict` in `QGparams()`:

```
G  <-
    flatten(
        apply(modelM2F[["VCV"]][ , grep("animal", colnames(modelM2F[["VCV"]]))],
          1,
          function(row) {
              list(matrix(row, ncol = 2))
          })
    )
R <-
```

```
    flatten(
        apply(modelM2F[["VCV"]][ , grep("units", colnames(modelM2F[["VCV"]]))],
              1,
              function(row) {
                  row[4] <- row[4] - 1
                  list(matrix(row, ncol = 2))
              })
    )
P <- map2(G, R, `+`)
```

Because we are using all the individual predictions, the computation takes a while, so just to illustrate here, we will only use the first MCMC iteration:

```
predict <- predict[1]
G <- G[1]
P <- P[1]
paramsM2F <-
    pmap(list(predict = predict,
              vcv.G = G,
              vcv.P = P),
         QGmvparams,
         models = c("Poisson.log", "binom1.probit"),
         verbose = FALSE)
```

6 Alternative software

While MCMCglmm is a great piece of software, there are various reasons for trying out different solutions, either because they are rather frequentist or using a different Bayesian framework. Here is a list of other R packages with which we can fit animal models:

brms A Bayesian package able to fit (non-linear) mixed models, based on the STAN framework. STAN is using a special case of MCMC, named Hamiltonian Monte Carlo (HMC), which requires a bit more time to compute each iteration, but results in much less auto-correlation in the output. Generally, this results in "faster" computation time for a given effective sample size. But, because it is less optimised than MCMCglmm for the particular use-case of animal models, there is no general rule to predict whether brms would be more or less efficient than MCMCglmm. There is a vignette on phylogenetic mixed model, which straightforwardly applies to animal model as well.

animalINLA Another Bayesian package, based on the R-INLA package. INLA (Integrated Nested Laplace Approximation) is a very fast (read "as fast as frequentist algorithms") way to approximate posterior distributions, which applies very well to linear mixed models (INLA notably specialises in spatial analysis). Animal-INLA can fit Gaussian, binomial and (zero-inflated) Poisson families. Of note: the last release was in 2016.

gremlin A frequentist and open source package for performing animal models using pedigrees. The syntax is relatively close to MCMCglmm. It can however only fit Gaussian responses. The package is still a work-in-progress with new features to come.

sommer Another frequentist, open-source package for performing animal models. The package is rather directed toward using marker-based relatedness matrices (rather than working with pedigrees). Just as gremlin, it can only fit Gaussian responses.

gaston Very similar package to sommer, but even more focused on fitting models using Genome-wide Relatedness Matrix (GRM) and other things related to quantitative genetics using genomic data.

pedigreemm A package to be able to provide a pedigree to lme4. No release since 2014.

7 Some references

Regarding quantitative genetics, and notably heritability estimation, three classical textbooks are very comprehensive (Falconer & Mackay 1996; Roff 1997; Lynch & Walsh 1998). They have completed by new references, including one focused on the quantitative genetics of wild populations (Charmantier et al. 2014) and a very thorough one on selection and response to selection (Walsh & Lynch 2018, a follow-up on the previous book by the same authors). A very substantial literature has developed around the quantitative genetics of wild populations: various reviews (Kruuk 2004; Postma & Charmantier 2007; Kruuk et al. 2008; Gienapp et al. 2017) and a guide (Wilson et al. 2010) are available on the animal model. Regarding MCMCglmm, the course notes by its developer Jarrod Hadfield is a really nice and accessible teaching document about mixed models and Bayesian statistics, as well as the package itself, of course. Some very nice and comprehensive books about Bayesian inference include Gelman et al. (2004) and the more recent and hands-on McElreath (2020).

References

Charmantier, A, Garant, D, & Kruuk, LEB, eds. (2014). Quantitative genetics in the wild. Oxford (UK): Oxford University Press. 293 pp. (cit. on p. 49).

Day, T & Bonduriansky, R (2011). A unified approach to the evolutionary consequences of genetic and nongenetic inheritance. *The American Naturalist*, 178, E18–E36 (cit. on p. 8).

Dempster, ER & Lerner, IM (1950). Heritability of threshold characters. *Genetics*, 35, 212–236 (cit. on p. 38).

de Villemereuil, P (2018). Quantitative genetic methods depending on the nature of the phenotypic trait. *Annals of the New York Academy of Sciences*. The Year in Evolutionary Biology 1422, 29–47 (cit. on pp. 3, 17, 33).

de Villemereuil, P, Gimenez, O, & Doligez, B (2013). Comparing parent-offspring regression with frequentist and Bayesian animal models to estimate heritability in wild populations: a simulation study for Gaussian and binary traits. *Methods in Ecology and Evolution*, 4, 260–275 (cit. on p. 34).

de Villemereuil, P et al. (2016). General methods for evolutionary quantitative genetic inference from generalised mixed models. *Genetics*, 204, 1281–1294. DOI: 10.1534/genetics.115.186536 (cit. on pp. 10, 32, 33).

de Villemereuil, P et al. (2018). Fixed-effect variance and the estimation of repeatabilities and heritabilities: issues and solutions. *Journal of Evolutionary Biology*, 31, 621–632. DOI: 10.1111/jeb.13232 (cit. on pp. 9, 26, 32).

Falconer, DS & Mackay, TF (1996). Introduction to quantitative genetics. 4th ed. Harlow, Essex (UK): Benjamin Cummings (cit. on pp. 5, 49).

Fisher, RA (1918). The correlation between relatives on the supposition of Mendelian inheritance. *Transactions of the Royal Society of Edinburgh*, 52, 399–433 (cit. on pp. 4, 11).

Gelman, A (2006). Prior distributions for variance parameters in hierarchical models. *Bayesian analysis*, 1, 515–533. DOI: 10.1214/06-BA117A (cit. on pp. 15, 16).

Gelman, A et al. (2004). Bayesian data analysis. Second. Text in Statisctical Science. Boca Raton, Florida (US): Chapman & Hall/CRC Press. 698 pp. (cit. on p. 49).

Gienapp, P et al. (2017). Genomic quantitative genetics to study evolution in the wild. *Trends in Ecology & Evolution*, 32, 897–908 (cit. on p. 49).

Hadfield, JD (2015). Increasing the efficiency of MCMC for hierarchical phylogenetic models of categorical traits using reduced mixed models. *Methods in Ecology and Evolution*, 6, 706–714 (cit. on pp. 34, 35).

— (2016). MCMCglmm Course Notes (cit. on pp. 3, 39).

Kirkpatrick, M & Lande, R (1989). The evolution of maternal characters. *Evolution*, 43, 485–503 (cit. on p. 8).

Kruuk, LEB (2004). Estimating genetic parameters in natural populations using the 'animal model'. *Philosophical Transactions of the Royal Society of London. Series B: Biological Sciences*, 359, 873–890 (cit. on p. 49).

Kruuk, LEB, Slate, J, & Wilson, AJ (2008). New answers for old questions: the evolutionary quantitative genetics of wild animal populations. *Annual Review of Ecology, Evolution, and Systematics*, 39, 525–548 (cit. on p. 49).

Lynch, M & Walsh, B (1998). Genetics and analysis of quantitative traits. Sunderland, Massachussets (US): Sinauer Associates (cit. on pp. 4, 5, 49).

McElreath, R (2020). Statistical Rethinking: A Bayesian Course with Examples in R and STAN. CRC Press. 575 pp. (cit. on p. 49).

Nakagawa, S & Schielzeth, H (2010). Repeatability for Gaussian and non Gaussian data: a practical guide for biologists. *Biological Reviews*, 85, 935–956 (cit. on pp. 10, 12, 33).

— (2013). A general and simple method for obtaining R2 from generalized linear mixed-effects models. *Methods in Ecology and Evolution*, 4, 133–142 (cit. on p. 26).

Postma, E & Charmantier, A (2007). What 'animal models' can and cannot tell ornithologists about the genetics of wild populations. *Journal of Ornithology*, 148, 633–642 (cit. on p. 49).

Räsänen, K & Kruuk, LEB (2007). Maternal effects and evolution at ecological time-scales. *Functional Ecology*, 21, 408–421 (cit. on p. 8).

Roff, DA (1997). Evolutionary quantitative genetics. New York (US): Chapman & Hall. 515 pp. (cit. on pp. 5, 49).

Walsh, B & Lynch, M (2018). Evolution and selection of quantitative traits. Oxford University Press. 1490 pp. (cit. on p. 49).

Wilson, AJ (2008). Why h2 does not always equal VA/VP? *Journal of Evolutionary Biology*, 21, 647–650 (cit. on pp. 9, 26).

Wilson, AJ et al. (2010). An ecologist's guide to the animal model. *Journal of Animal Ecology*, 79, 13–26 (cit. on pp. 8, 49).

Wolak, ME (2012). Nadiv : an R package to create relatedness matrices for estimating non-additive genetic variances in animal models. *Methods in Ecology and Evolution*, 3, 792–796 (cit. on p. 27).

Wolak, ME & Reid, JM (2017). Accounting for genetic differences among unknown parents in microevolutionary studies: how to include genetic groups in quantitative genetic animal models. *Journal of Animal Ecology*, 86, 7–20 (cit. on p. 6).

Wright, S (1934). An analysis of variability in number of digits in an inbred strain of Guinea pigs. *Genetics*, 19, 506–536 (cit. on p. 33).