How to use the QGglmm package?

Pierre de Villemereuil

September 18, 2024

Contents

1 Summary and aim of the package **2**

2 Overview of the theory **2**

2.1 Quantitative genetics and linear mixed models 2

2.2 Generalised linear mixed model 3

2.3 An illustrative example 4

2.4 Quantitative genetics and GLMM 7

3 The framework behind QGglmm **9**

3.1 Computing the phenotypic mean and variance 9

3.2 Computing the additive genetic variance and related parameters 10

3.3 The issue of fixed effects 11

4 Using QGglmm **11**

4.1 Extracting information from the models 11

4.2 Using QGparams to obtain parameters on the observed data scale 13

4.3 Using QGmvparams to obtain multivariate parameters on the observed data scale 17

4.4 Using QGpredict to obtain evolutionary prediction 20

5 Particular cases **22**

5.1 Including fixed effects 22

5.2 Obtaining intra-class correlation (ICC) coefficients 24

5.3 Integrating over a posterior prediction 25

5.4 The special case of ordinal data 27

1 Summary and aim of the package

The QGglmm package is an R implementation of the framework developed by de Villemereuil *et al.* (2016) to compute quantitative genetics parameters on the observed data scale after a Generalised Linear Mixed Model (GLMM) analysis. It allows for the computation of the mean, variances and heritability on the observed data scale, as well as for evolutionary predictions if measures of fitness gradient are provided. For a comprehensive description of the framework, please read de Villemereuil *et al.* (2016).

The package is meant to be used *after* an inference from a GLMM. As a consequence, *the package does not perform any inference*. To infer genetic additive variances from your experimental design, please refer to packages or software such as MCMCglmm or ASreml.

This "How-to" is destined for people having performed a quantitative analysis through a GLMM and wanting to use QGglmm to obtain quantitative genetic parameters (e.g. additive genetic variance, heritability or G matrix) on the observed data scale rather than the latent scale (see below for a definition of the scales). It also explain how to use the package to perform evolutionary predictions if fitness information is available.

2 Overview of the theory

2.1 Quantitative genetics and linear mixed models

Nowadays, the reference kind of model for performing quantitative genetics analysis is the linear mixed model (LMM), and especially a particular form of called the animal model (Henderson, 1950, 1976; Kruuk, 2004), whereby the additive genetic variance of the trait is estimated using relatedness information between individuals. Mathematically, a linear mixed model of a phenotypic trait $\mathbf{z}$[1] is of the following shape:

$$\mathbf{z} = \mu + \mathbf{Xb} + \mathbf{Z_a a} + \mathbf{Z_1 u_1} + ... + \mathbf{Z}_k \mathbf{u_k} + \mathbf{e}, \tag{1}$$

where μ is the intercept of the model, $\mathbf{X}$ is called the design matrix and contains the fixed effect co-variates and $\mathbf{b}$ contains the estimated parameters for the fixed effects. The random effects are separated between the additive genetic component $\mathbf{Z_a a}$ and other possible random effects $\mathbf{Z_1 u_1} + ... + \mathbf{Z}_k \mathbf{u_k}$. Finally, $\mathbf{e}$ is the residual component (i.e. the "error" term).

[1] $\mathbf{z}$ is a vector containing the trait value for each individual, hence its length is equal to the number of individuals in the study.

There are several key assumptions in a LMM, among which are:

- The residual $\mathbf{e}$ follows a Normal distribution with independent effects between individuals:

$$\mathbf{e} \sim \mathcal{N}(\mathbf{0}, \mathbf{I}V_{\mathrm{R}}) \tag{2}$$

  where $\mathbf{I}$ is the identity matrix and V_{R} is the residual variance.

- The additive genetic component (the "breeding values") are normally distributed with a variance-covariance matrix $\mathbf{A}$ being composed of the relatedness between individuals:

$$\mathbf{a} \sim N\left(\mathbf{0}, \mathbf{A}V_{\mathrm{A},\ell}\right), \tag{3}$$

  where $V_{\mathrm{A},\ell}$ is the additive genetic variance.

- All random effects are independent from each other and from the fixed effects.

This model is very well supported by quantitative genetics. According to Fisher (1918)'s infinitesimal model, breeding values do indeed follows Eq. 3. Also, that the combined result of genetical and environmental effects follows a Normal distribution is a classical assumptions in quantitative genetics. Because of that, most tools and theory in quantitative genetics assume this kind of distribution.

2.2 Generalised linear mixed model

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.

This structure consists of three "scales" depicted in Fig. 1 and which can be written using the following equations:

$$\boldsymbol{\ell} = \mu + \mathbf{Xb} + \mathbf{Z_a a} + \mathbf{Z_1 u_1} + ... + \mathbf{Z}_k\mathbf{u_k} + \mathbf{o}, \tag{4a}$$

$$\boldsymbol{\eta} = g^{-1}(\boldsymbol{\ell}), \tag{4b}$$

$$\mathbf{z} \sim \mathcal{D}(\boldsymbol{\eta}, \boldsymbol{\theta}), \tag{4c}$$

Eq. 4a refers to the latent trait $\boldsymbol{\ell}$.

By comparing it to Eq. 1, we can see that the same assumptions are made for $\boldsymbol{\ell}$ 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.

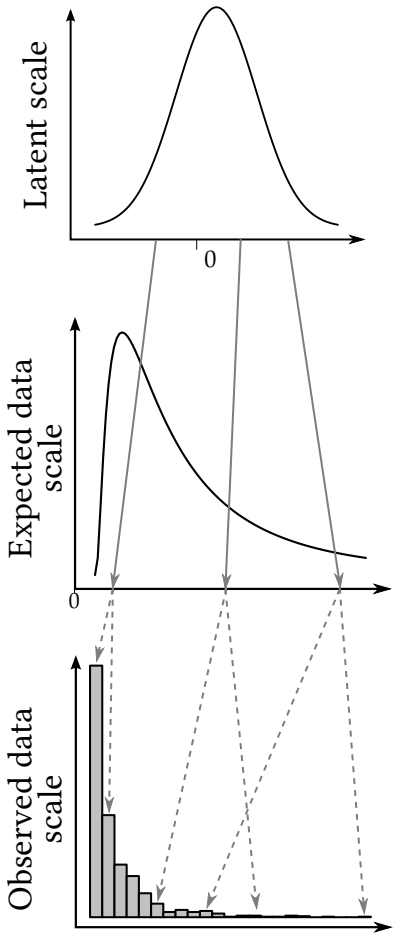

Figure 1: The three scales of the Generalised Linear Mixed Model (here using Poisson with a log link function as an example). The error terms are normally distributed on the latent scale, but follows a Poisson distribution on the observed data scale (conditionnally on the latent scale).

In de Villemereuil *et al.* (2016), Eq. 4b is said to refer to the "expected data scale". This is because the term $\boldsymbol{\eta}$ is the individual expectation around which the observed data are realised (see example below). The transition from the latent scale to the expected data scale is preformed by the inverse of the link-function g. 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 binomial distribution can only use values between 0 and 1, hence the use of logit or probit link-function which match these input and output realms.
Finally, Eq. 4c models the "observed data scale" by adding, to the expectation $\boldsymbol{\eta}$, an error term from a given non-Normal distribution (here noted $\mathcal{D}$), which can accept additional parameters $\boldsymbol{\theta}$.

It is very important to realise that 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 $V_{A,\ell}$ 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 2.4).

2.3 An illustrative example

All the statistical soup above might seem difficult to digest to some readers. In order to best explain some of these technical facts, we will use an illustrative example directly simulated in R. This example will use a Poisson with a log link-function GLMM.

Let's consider a very simple model for now, with no fixed effects and no random effects except for the additive genetic one. Following Eqs. 4, we can write the model as:

$$\boldsymbol{\ell} = \mu + \mathbf{Z}_\mathbf{a}\mathbf{a} + \mathbf{o}, \tag{5a}$$

$$\boldsymbol{\eta} = \exp(\boldsymbol{\ell}), \tag{5b}$$

$$\mathbf{z} \sim \mathcal{P}(\boldsymbol{\eta}), \tag{5c}$$

Note that, because we use the inverse of the log link-function, we use, well, the inverse of the logarithm, which is an exponential. Note that a Poisson can only use positive real number as input and that exponential only yields positive real numbers, so the "matching" performed by the link-function is quite obvious here.

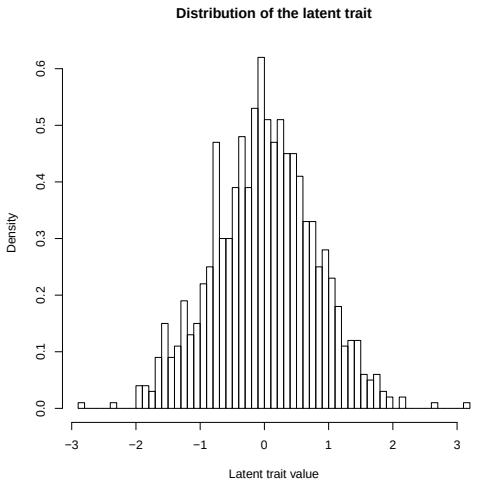

Figure 2: Distribution of the simulated latent trait.

So, our model has three parameters: the intercept μ, the additive genetic variance $V_{\mathrm{A},\ell}$ and the over-dispersion variance V_{O}. Let's define some values for them:

```
# Value for the intercept mu
mu <- 0
# Additive genetic variance
Va <- 0.3
# Over-dispersion variance
Vo <- 0.3
# Number of individuals
N  <- 1000
```

Using these parameters, we can simulate the latent trait:

```
l <- mu + rnorm(N, 0, sqrt(Va)) + rnorm(N, 0, sqrt(Vo))
```

The distribution of the simulated latent trait is illustrated in Fig. 2[2].

Now that we have constructed our latent trait, we can use the inverse of the link-function to compute the values on the expected data scale (a.k.a. $\boldsymbol{\eta}$):

```
eta <- exp(l)
```

Yes, it's that simple. The distribution of $\boldsymbol{\eta}$ is illustrated in Fig. 3. From this graph and the fact that we are simply taking the exponential of the latent trait, we can see that two things are going to happen: much of the values are going to be low (say between 0 and 1), whereas a few (corresponding to the upper tail of the Normal distribution) are going to yield large values (in Fig. 3, we see the extreme is just above 20). This is all good and well, but doesn't quite really illustrate what $\boldsymbol{\eta}$ is biologically. The best way to see this is to compute the observed phenotypic trait from it.

[2]Note that because we do not have fixed effects, the distribution is clearly Gaussian, but when fixed effects are included, there is no expectation for the distribution of $\boldsymbol{\ell}$ to be Gaussian (only the residual is, i.e. when all effects have been "removed").

The observed phenotypic trait $\mathbf{z}$ is the realisation around the individual expectation $\boldsymbol{\eta}$ according to the chosen distribution $\mathcal{D}$. In our case, what this means is that for individual i, its phenotype z_i is drawn from a Poisson distribution with a mean (i.e. the famous λ parameter of the Poisson distribution) η_i:

```
z <- rpois(N, lambda = eta)
```

Now, the distribution of our simulated phenotypic trait is illustrated in Fig. 4. We can see that this distribution is heavily non-Gaussian[3], with a lot of 0 and a few large values (again up to slightly above 20).

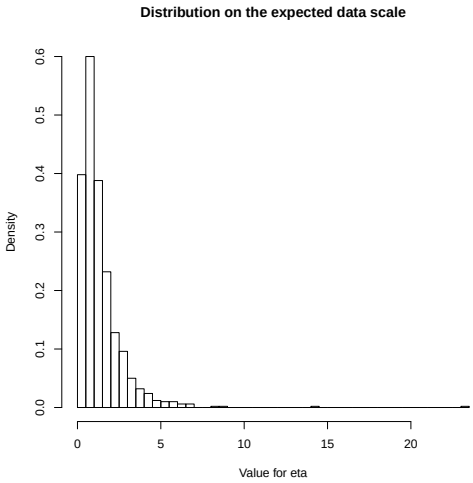

Figure 3: Distribution of the simulated $\boldsymbol{\eta}$ values.

To fully realise the meaning of the different scales, it can be interesting to "follow" the values for a particular individual, say the 101th. First, we can have a look at its latent value:

```
l[101]
eta[101]
z[101]
```

```
1.225813
3.406933
4
```

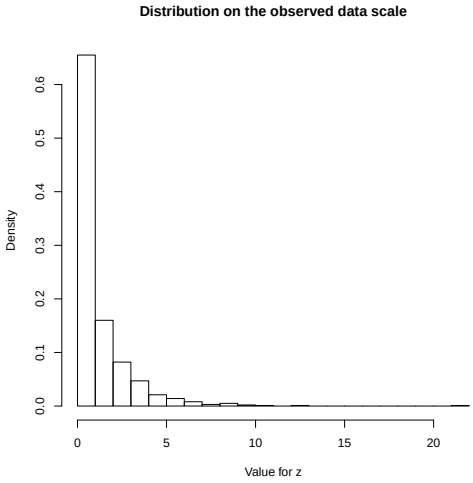

Figure 4: Distribution of the simulated z values.

So, this individual have a latent trait value of 1.23. This value can be directly compared to our value of the population latent mean μ, which was 0. This individual's latent component tends thus toward bigger values than the average. As a result, its expected value is of 3.4. Let's say that our trait is the mating success of a population of male. This means that this individual is expected, according to its latent (i.e. in part, according to its genetics), to mate with 3.4. Now, we all know that it's impossible to mate with 3.4 females: we can only observe integer values of mating success. This is the observed value, which, in our case was 4, but could have been anything e.g. between 1 and 7:

```
qpois(0.05, lambda = 3.4)
qpois(0.95, lambda = 3.4)
```

[3]At this point, we need to remind the reader that we are authorised to look at the overall shape of the different scales directly because our simulations did not have any fixed effects. A variable can totally have a weird distribution, but a normally distributed error term if the correct covariate is used as a fixed effect.

```
1
7
```

It is thus important to realise that the observed values can be quite different from the expected ones and, more importantly, that the noise assumed on this latter part of the model can be very large. We will come to this latter, when mentioning the different values of the heritability.

This whole simulation is basically the process assumed by a GLMM. Latent values are assigned, according to all predictors (here only an additive genetic one), to all individuals, or observations if they are repeated measurements. These latent values are transformed, using the inverse-link-function into expected values, around which observed values are drawn. The main difference when performing a statistical inference with a GLMM is that only observed values are known, the rest is entirely inferred by the model.

It would be advisable, for anyone using a GLMM, to toy around with these kind of simulations (changing the parameters and distribution to fit the case at hand), in order to get familiar with the implications of some parameters values. Most of the time, indeed, change in a parameter value will have quite different consequences on the data scale compared to what one can expect in a LMM. This is illustrated from the simulation above. Note that the latent mean is 0, yet all simulated data have a positive value. Note also the low values for the latent variances, whereas the simulated data have a much larger variance.

2.4 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. Much of what will be mentioned here comes from de Villemereuil *et al.* (2016), so we will just summarise some issues and try to keep it simple.

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 same 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*[4] that at some point, *something* is normally distributed. We simply call this something "latent trait".

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

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 where observed values are drawn around the expected values following the distribution $\mathcal{D}$. 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 be some variance, depending on the value of the trait.
The second source is the inverse-link-function. It is not *creating* noise (it's a function, not a statistical 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[5].
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 distribution $\mathcal{D}$, heritability on the observed data scale can never reach a maximum 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 QGglmm is about. But this has further consequences. Narrow-sense heritability is nothing like repeatability[6] when they are computed on the observed data scale. This means that some of the computations and advice available in Nakagawa & Schielzeth (2010) are not applicable to narrow-sense heritability on the observed scale. However, broad-sense heritability (i.e. including the non-additive effects) can

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

[6]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.

be computed as a repeatability-like estimate (a.k.a. intra-class correlation coefficients, ICC). QGglmm also provides a way to compute ICCs, even for non-genetic components, see section 5.2.

3 The framework behind QGglmm

Most of the content of this section is explained in more details in the article introducing the framework (de Villemereuil *et al.*, 2016). We will here just have an overview of the computations involved, as they are needed to understand what the package is doing.

Exactly knowing how to compute the parameters is, of course, QGglmm's business. It will start computation with the correct functions given the distribution name it has been given (see the practice in the following section). Interpreting the output of QGglmm however is up to the user, and having an idea of what happens behind the scene of a package is always the best strategy for a correct interpretation.

3.1 Computing the phenotypic mean and variance

The first parameters to compute are the population phenotypic mean and variance on the observed data scales.

Population mean The last layer, going from Eq. 4b to Eq. 4c just adds noise around some expected values (computed as $g^{-1}(\ell)$). Hence, the population mean is the average of these expected values:

$$\bar{z} = \int g^{-1}(\ell) f_{\mathcal{N}}(\ell, \mu, V_{\mathrm{P},\ell}) \mathrm{d}\ell, \tag{6}$$

where $f_{\mathcal{N}}(\ell, \mu, V_{\mathrm{P},\ell})$ is the probability density of a Normal distribution with mean μ and variance $V_{\mathrm{P},\ell}$ evaluated at ℓ. Note that $V_{\mathrm{P},\ell}$ is the sum of all variance components (i.e. $V_{\mathrm{A},\ell}$, $V_{\mathrm{O},\ell}$ and random effect variances) on the latent scale. In other words, it is the "phenotypic variance" of the latent trait.

Variance on the expected data scale The variance on the expected data scale is simply the variance of the above-mentioned expected values around the population mean:

$$V_{\mathrm{P,exp}} = \int \left(g^{-1}(\ell) - \bar{z}\right)^2 f_{\mathcal{N}}(\ell, \mu, V_{\mathrm{P},\ell}) \mathrm{d}\ell. \tag{7}$$

Variance on the observed data scale To compute the variance of the observed data scale, we need to the noise arising when going from Eq. 4b to Eq. 4c. We call this variance the "distribution variance" and it depends on a variance function $v(\ell, \boldsymbol{\theta})$ that describe the variance

of the distribution $\mathcal{D}$ used[7]:

$$V_{\text{dist}} = \int v(\ell, \boldsymbol{\theta}) f_{\mathcal{N}}(\ell, \mu, V_{\text{P},\ell}) \mathrm{d}\ell. \tag{8}$$

To compute the phenotypic variance on the observed data scale, we simply need to add Eqs. 7 and 8:

$$V_{\text{P,obs}} = V_{\text{P,exp}} + V_{\text{dist}}. \tag{9}$$

We mentioned earlier that GLMM were quite noisy models. We can see here why: on the observed data scale, we have variance arising from the latent scale that went through the link function ($V_{\text{P,exp}}$) to which we further add the distribution variance (V_{dist}).

3.2 Computing the additive genetic variance and related parameters

To compute the additive genetic variance on the observed data scale $V_{\text{A,obs}}$ from the latent scale, two things are needed: the latent additive genetic variance $V_{\text{A},\ell}$ and a parameter called Ψ[8]. The details of Ψ computation can be omitted here, but what is important to note is that, as all computations above, it depends on the whole distribution of the latent trait ℓ. $V_{\text{A,obs}}$ can then be computed as:

$$V_{\text{A,obs}} = \Psi^2 V_{\text{A},\ell} \tag{10}$$

Simple enough (if you have Ψ!).

Once the value for $V_{\text{A,obs}}$ and all the other parameters have been obtained, it is pretty straightforward to compute other related parameters on the observed data scale, such as the heritability:

$$h^2_{\text{obs}} = \frac{V_{\text{A,obs}}}{V_{\text{P,obs}}}, \tag{11}$$

or the coefficient of variation:

$$\text{CV}_{\text{A,obs}} = 100 \frac{\sqrt{V_{\text{A,obs}}}}{\bar{z}}, \tag{12}$$

or even the closely-related evolvability:

$$\text{I}_{\text{A,obs}} = \frac{V_{\text{A,obs}}}{\bar{z}^2}. \tag{13}$$

[7] For a Binomial distribution, for example, it is equal to $g^{-1}(\ell)\left(1 - g^{-1}(\ell)\right)$, similar to the classical $p(1-p)$.

[8] This parameter can be considered as the slope of the linear regression from the latent breeding values to the breeding values on the observed data scale. For more details, see de Villemereuil *et al.* (2016)

3.3 The issue of fixed effects

Fixed-effects can be messy in GLMMs because of the non-linearity introduced by the link function. Indeed, when fixed-effects are introduced in the model[9], they tend to have a strong effect on the shape of the latent trait distribution. Yet, you might remember that all computations describe in Eqs. 6–10 depends ultimately on the whole distribution of the latent trait.

The immediate consequence of this is that include fixed effects will have a noticeable and possibly large impact on the estimates on the observed data scale. To account for that, it is necessary to average over fixed-effects (see de Villemereuil *et al.*, 2016, for more details). QGglmm is able to do that and we will see how in the practice. However, it should be noted that averaging over fixed-effects can be computationally demanding, especially for large datasets, because the computation time grows roughly linearly with the sample size (i.e. computations needs to be done for *each* data point). See section 5.1 for more information about this.

4 Using QGglmm

4.1 Extracting information from the models

As explained at the beginning of this document, QGglmm does not perform any kind of inference directly from the data. Instead, estimates from the latent scale (e.g. intercept, total variance and additive genetic variance) are its input. This means that the package will usually be used *after* some analysis using a GLMM-based software has been conducted.

It is thus up to the user to extract these estimates from the software output and provide them to QGglmm. No automatic extraction of these estimates directly from model objects is available. This is mainly for two reasons:

- There are a very large diversity of GLMM-based software out there, which makes difficult the task to implement an automatic extraction for all of them,
- But most importantly, every statistical design is unique, and automatic extractions can only very roughly account for them. What if the user wants to discard some random effect variance from the total variance? Which variance component is the additive genetic one (e.g. for sire/dam models)? Etc...

As an illustration, we will however see how to extract estimate values from different but simple models using MCMCglmm and lme4.

[9]Others than the bare intercept μ, which, for the sake of simplicity, we do not include in what we call "fixed-effects" here.

Animal model with MCMCglmm Imagine we have an animal model fitted with MCMCglmm to analyse count data using a Poisson distribution. The model would be fitted using, for example, the following command:

```
model <- MCMCglmm(phen ~ 1, random = ~animal,
                  pedigree = pedigree, data = data,
                  prior = prior, family = "poisson")
```

Now, all we need is to extract, from the `model` object, the latent population mean (here, the intercept of the model), additive genetic variance and phenotypic variance:

```
mu <- mean(model[["Sol"]][ , "(Intercept)"])
va <- mean(model[["VCV"]][ , "animal"])
vp <- mean(rowSums(model[["VCV"]]))
```

And that's everything that will be needed by QGglmm. For binary traits, it is not needed to add 1 (for probit link) or $\pi^2/3$ (for logit link)

Multivariate animal model with MCMCglmm Now, what if we wanted to fit a multivariate animal model rather than a univariate one? Something like this:

```
model <- MCMCglmm(cbind(phen1, phen2) ~ trait - 1,
                  random = ~ us(trait):animal,
                  rcov = ~us(trait):units,
                  data = data, pedigree = pedigree,
                  family = c("ordinal", "gaussian"),
                  prior = prior)
```

Extracting the parameters from the `model` object is a bit more tedious here, because most of them are actually matrices and need to be re-formatted as such:

```
mu <-  c(mean(model[["Sol"]][ , "traitphen1"]),
         mean(model[["Sol"]][ , "traitphen2"]))
G <-
  matrix(c(mean(model[["VCV"]][ , "traitphen1:traitphen1.animal"]),
           mean(model[["VCV"]][ , "traitphen1:traitphen2.animal"]),
           mean(model[["VCV"]][ , "traitphen1:traitphen2.animal"]),
           mean(model[["VCV"]][ , "traitphen2:traitphen2.animal"])),
         ncol = 2)
R <-
  matrix(c(mean(model[["VCV"]][ , "traitphen1:traitphen1.units"]),
           mean(model[["VCV"]][ , "traitphen1:traitphen2.units"]),
           mean(model[["VCV"]][ , "traitphen1:traitphen2.units"]),
           mean(model[["VCV"]][ , "traitphen2:traitphen2.units"])),
         ncol = 2)
P <- G + R
```

Note that contrary to the command lines on the previous paragraph, those lines are quite specific to the model used: they use the name of the phenotypic variables (`phen1` and `phen2`) and the last command assumes that no random effect was included in the model. Should there be more random effects, their variance-covariance matrix of course need to be added up to obtain the phenotypic variance-covariance matrix `P`.

Sire/dam model with lme4 Imagine now we have a sire/dam model fitted with lme4 to analyse binary data[10]. The model would be fitted using, for example, the following command:

```
model <- glmer(phen ~ 1 + (1|sire) + (1|dam),
         data = data, family = binomial(link = "probit"))
```

What we would like is to extract the intercept and variance components from the `model` object. This can be done using the following commands:

```
vars <- as.data.frame(VarCorr(model))[ , c('grp', 'vcov')]
intercept <- fixef(model)['(Intercept)']
```

Now, all is needed left is to construct the actual parameters of interest. The latent population mean is quite straightforward:

```
mu <- intercept
```

The additive genetic variance of the model is four times the sire variance and the latent phenotypic variance on the latent scale is simply the sum of the variance components:

```
va <- 4 * vars[vars[["grp"]] == "sire", "vcov"]
vp <- sum(vars[, "vcov"])
```

And just like that, we have everything required to use QGglmm!

4.2 Using QGparams to obtain parameters on the observed data scale

Getting some parameters to play with Let's assume we performed an analysis using any kind of GLMM software on a count trait. From this analysis, we obtain the following parameters:

```
va <- 1.34
vp <- 2.13
mu <- 0.89
```

[10]Note that the model used here belongs to an analysis assuming that "dam effect" contains maternal effects but "sire effect" doesn't contain paternal effects.

Running QGparams (the easy way) We can use these parameters and our knowledge of the model (here a Poisson model with a log link) to compute the quantitative genetics parameters on the observed data scale using the simple following command:

```
QGparams(mu = mu, var.a = va, var.p = vp, model = "Poisson.log")
```

```
[1] "Using the closed forms for a Poisson-log model."
  mean.obs  var.obs var.a.obs    h2.obs
1 4.116486 35.59524  9.150549 0.2570723
```

From this output, we can see different parameters on the observed phenotypic trait, i.e. the count number:

- The population mean `mean.obs` of roughly of 4.1, meaning this is the average expected count number in the population.
- The variance `var.obs` is the total phenotypic variance of the count number in the population. It's large value (35.6) is quite typical for a Poisson/log distributed trait.
- The variance `var.a.obs` is the additive genetic variance of the count data.
- The parameter `h2.obs` is the ratio between `var.a.obs` and `var.obs`, i.e. the heritability of the count number.

If we compare the value of the latent heritability:

```
va/vp
```

```
0.5142857
```

with the value of the heritability on the observed data scale (here 0.26), we can see that they differ greatly, and that the latter is smaller than the former. Though the absolute difference can vary depending on the data and model, `h2.obs` is *always* expected to be smaller than the latent heritability. This is due to the added "noise" in GLMM models (see section 2.4).
Because the Poisson/log model has a closed-form solution to our framework, the computation is very quick. Some common models (e.g. binomial with logit link) do not have such solution (yet?) and for them computation is slower. You can see whether `QGparams` is using the closed forms through its verbose output:

```
QGparams(mu = mu, var.a = va, var.p = vp,
         model = "Poisson.log", closed.form = FALSE)
```

Table 1: Name, description and characteristics of the models implemented in QGglmm.

Name	Description	Closed form?	Extra parameter	Comment
`Gaussian`	Gaussian distribution with identity link	✓	—	Essentially: LMM
`binom1.probit`	Binomial with 1 trial and probit link	✓	—	For binary trait
`binomN.probit`	Binomial with N trials and probit link	✓	`n.obs`	`n.obs` is the number of trials
`binom1.logit`	Binomial with 1 trial and logit link	✗	—	For binary trait
`binomN.logit`	Binomial with N trials and logit link	✗	`n.obs`	`n.obs` is the number of trials
`Poisson.log`	Poisson distribution wiht a log link	✓	—	—
`Poisson.sqrt`	Poisson distribution with a square-root link	✓	—	—
`negbin.log`	Negative-Binomial distribution wiht a log link	✓	`theta`	`theta` is the overdispersion parameter
`negbin.sqrt`	Negative-Binomial distribution wiht a square-root link	✓	`theta`	`theta` is the overdispersion parameter
`ordinal`	Multiple threshold model for ordinal traits	✓	—	See section 5.4.

```
[1] "Computing observed mean..."
[1] "Computing variances..."
[1] "Computing Psi..."
  mean.obs  var.obs var.a.obs    h2.obs
1 4.116486 35.59524  9.150549 0.2570723
```

Note the longer and more verbose output than latter. Here, `QGparams` takes time (not MUCH more, but still more) to compute integrals for the mean, then variances, then Psi...
A last thing to note is the format of the output: `QGparams` yields one row of a `data.frame`, this will become handy in section 5.3 on posterior distribution, because those rows are easily stackable.

List of available models There are quite a bunch of models that are implemented in QGglmm, for which we have coded all the functions needed for computation and the closed forms when available. When available, `QGparams` defaults to using the closed forms (as can be seen in the above paragraph). The table 1 lists all the currently available models and some of their characteristics.

Running QGglmm (the custom way) What should be done if the model you used is not in this list? Well, that means that using QGglmm is still possible but will require somewhat more efforts from *your* part. Of course, if you don't need to implement your own model, you can skip this rather technical part[11]. Three things are needed:

[11]Though I would recommend readers to read it, as it explains how `QGparams` works "under the hood".

`inv.link` The inverse function of the link function your model, i.e. if you model uses a link function g, you need to impute g^{-1}.

`var.func` The "variance function" $v(\ell, \boldsymbol{\theta})$ (see Eq. 8), which is the function giving the variance to expect around a value for the latent trait ℓ given some (possibly empty!) parameter $\boldsymbol{\theta}$, assuming your custom distribution $\mathcal{D}$.

`d.inv.link` Derivative of the inverse-link function `inv.link`, this should require little mathematical work since link functions are often relatively simple.

To illustrate how to compute and use those things, we will "manually" implement the binomial/logit model. So, the first thing is to know what the link function looks like. Wikipedia tells us that a logit function looks like this:

$$g(x) = \log\left(\frac{x}{1-x}\right) \tag{14}$$

The inverse of this function is so that:

$$g(g^{-1}(x)) = x = \log\left(\frac{g^{-1}(x)}{1-g^{-1}(x)}\right) \tag{15}$$

A tiny bit of mathematical work gives:

$$g^{-1}(x) = \frac{\exp(x)}{1+\exp(x)} \tag{16}$$

This, is our `inv.link` function:

```
inv.link <- function(x){exp(x)/(1+exp(x))}
```

Derivating this function is quite easy and gives:

$$\frac{\mathrm{d}g^{-1}(x)}{\mathrm{d}x} = \frac{\exp(x)}{(1+\exp(x))^2} \tag{17}$$

So, `d.inv.link` is:

```
d.inv.link <- function(x){exp(x)/((1+exp(x))^2)}
```

Now, the last bit requires more thinking. The function $v(x, \boldsymbol{\theta})$ gives us the variance of the values on the observed data scale for a given latent value x. So, for a given latent value x, Eq. 4 tells that the distribution of the values will be:

$$z|x \sim \mathcal{B}(g^{-1}(x), 1) \tag{18}$$

where $\mathcal{B}(p, N)$ is the binomial distribution with probability of success p and number of trials N (here $N = 1$ is our general parameter $\boldsymbol{\theta}$). We know that the variance of this distribution is

generally $Np(1-p)$, so in our case:

$$v(x, \theta) = g^{-1}(x)(1 - g^{-1}(x)) \tag{19}$$

So, we finally have `var.func`:

```
var.func <- function(x){(exp(x)/(1+exp(x))) *
                        (1 - (exp(x)/(1+exp(x))))}
```

QGglmm requires that these functions are provided using a named list:

```
custom.functions <- list(inv.link = inv.link,
                         var.func = var.func,
                         d.inv.link = d.inv.link)
```

Now, we can use our "custom" model in QGparams:

```
QGparams(mu = 0.5, var.a = 0.5, var.p = 1,
         custom.model = custom.functions)
```

```
[1] "Computing observed mean..."
[1] "Computing variances..."
[1] "Computing Psi..."
   mean.obs   var.obs var.a.obs     h2.obs
1 0.6020271 0.2395905 0.0197978 0.08263184
```

which gives exactly the same output as the built-in model:

```
QGparams(mu = 0.5, var.a = 0.5, var.p = 1, model = "binom1.logit")
```

```
[1] "Computing observed mean..."
[1] "Computing variances..."
[1] "Computing Psi..."
   mean.obs   var.obs var.a.obs     h2.obs
1 0.6020271 0.2395905 0.0197978 0.08263184
```

Surprisingly, QGglmm does not need to *know* exactly which distribution you are using, and the only part where the distribution $\mathcal{D}$ (and actually, solely its variance) is used is for the computation of `var.func`.

4.3 Using QGmvparams to obtain multivariate parameters on the observed data scale

Getting some parameters to play with Let's assume we just performed a large multivariate quantitative genetics analysis with 3 traits, among which a binary trait (binomial/probit), a Gaussian trait and a count trait (Poisson/log). We obtained the following parameters:

```
G <- matrix(c(0.5, 0.4, 0, 0.4, 2, 0, 0, 0, 0.1), nrow = 3)
P <- matrix(c(1, 0.4, 0, 0.4, 5, 0, 0, 0, 0.5), nrow = 3)
mu <- c(-0.5, 10, 1)
```

By looking at G, we can that the two first traits are genetically correlated, but the last one is independent from the others (on the latent scale!):

```
G
```

```
     [, 1] [, 2] [, 3]
[1, ]  0.5  0.4  0.0
[2, ]  0.4  2.0  0.0
[3, ]  0.0  0.0  0.1
```

All the phenotypic covariance is of phenotypic origin though:

```
  P - G
```

```
     [, 1] [, 2] [, 3]
[1, ]  0.5    0  0.0
[2, ]  0.0    3  0.0
[3, ]  0.0    0  0.4
```

If only real estimates could be that clean!

Running QGmvparams The function `QGmvparams` behave mostly the same as `QGparams`, but has a modified arguments and output to adapt to the multivariate case. For example, it requires not simply variances, but variance-covariance matrices and a vector of intercept for `mu`. It also requires that all the models used for the trait are given (as a vector):

```
QGmvparams(mu = mu, vcv.G = G, vcv.P = P,
           model = c("binom1.probit", "Gaussian", "Poisson.log"))
```

```
[1] "Computing observed mean..."
[1] "Computing variance-covariance matrix..."
[1] "Computing Psi..."
$mean.obs
[1]  0.361887 10.000149  3.490865

$vcv.P.obs
              [, 1]      [, 2]          [, 3]
[1, ]  0.2308944225 0.1056209 -0.0002702066
[2, ]  0.1056208909 5.2382325  0.0577576973
[3, ] -0.0002702066 0.0577577 11.3775643645

$vcv.G.obs
```

```
           [, 1]      [, 2]     [, 3]
[1, ] 0.03511883 0.1063377 0.000000
[2, ] 0.10633770 2.0124007 0.000000
[3, ] 0.00000000 0.0000000 1.225777
```

A few things ought to be noted here regarding this output. First, you can see that the output has substantially changed. It is not a `data.frame` any more, but a `list` object. Not as easily stackable as `data.frame`, but much more convenient to store matrices! Second, despite the fact that all models used here have a closed form (see Table 1), `QGmvparams` seems to be doing the integral computation (see the verbose output on the three first lines). This is because the computation for each model are not independent, so using closed forms would require to solve this particular case (binomial/probit + Gaussian + Poisson/log). This is much more complex than univariate models and there are so many possibilities that using closed forms for the multivariate is practically impossible. Hence `QGmvparams` *always* resort to integral computations (using the cubature package)[12].

Looking closer at the estimates This example is interesting because it allows to see some of the peculiar stuff happening when converting from the latent to the observed data scale. Let's save the output of `QGmvparams` to study it further:

```
out <- QGmvparams(mu = mu, vcv.G = G, vcv.P = P,
       model = c("binom1.probit", "Gaussian", "Poisson.log"))
```

One interesting thing is to compare the additive genetic variance on the latent scale and on the observed data scale:

```
G
out[["vcv.G.obs"]]
```

```
     [, 1] [, 2] [, 3]
[1, ]  0.5  0.4  0.0
[2, ]  0.4  2.0  0.0
[3, ]  0.0  0.0  0.1

           [, 1]      [, 2]     [, 3]
[1, ] 0.03511883 0.1063377 0.000000
[2, ] 0.10633770 2.0124007 0.000000
[3, ] 0.00000000 0.0000000 1.225777
```

[12] In previous versions, QGglmm was using R2Cuba which has been deprecated. The switch to cubature, however, allowed for using its "vectorised" approach which considerably improved the speed of the package: up to 50x for QGmvparams and up to x300 faster for QGmvicc!!

You can see that some of the structure was kept during the transformation. For example, the last trait is still genetically independent from the others. The estimates regarding the Gaussian haven't changed much (this is because the Gaussian trait is not really "transformed", it is pretty much kept the same). Yet, the variances of the non-Gaussian traits changed: the first one was reduced, while the third one was increased.
The changes about the non-genetic part are more striking:

```
P
out[["vcv.P.obs"]]
```

```
     [, 1] [, 2] [, 3]
[1, ]  0.5    0  0.0
[2, ]  0.0    3  0.0
[3, ]  0.0    0  0.4

               [, 1]           [, 2]           [, 3]
[1, ]  0.1957755888 -0.0007168094 -0.0002702066
[2, ] -0.0007168094  3.2258318034  0.0577576973
[3, ] -0.0002702066  0.0577576973 10.1517870576
```

whereas the variance-covariance of the non-genetic part on the latent scale is completely diagonal, this is not the case any more on the observed data scale: the second and third traits now slightly covariate whereas they were independent on the latent and are not even genetically correlated. This happens because of the non-linearity of the link functions (again!) which can generate a slight covariance between traits, even if they are not correlated on the latent scale.

4.4 Using QGpredict to obtain evolutionary prediction

Why use QGpredict? Obtaining heritability estimates on the observed data scale is interesting because it allows to infer how of the variance of the *actual* phenotypic trait is of additive genetic origin. Yet, contrary to what happens for Gaussian traits, heritability of non-Gaussian traits is a poor predictor of the evolutionary response to selection (de Villemereuil *et al.*, 2016). This is where `QGpredict` is useful: using fitness information on the observed data scale, it allows to infer evolutionary predictions on the latent scale.

How does it work? The starting point of the approach is to have fitness information for each phenotypic category (non-Gaussian traits are most often categorical traits: dichotomies, counts, ordinal traits, etc...), or a fitness model which allows to compute the expected fitness given a latent value ℓ:

$$W_{\text{exp}}(\ell) = \sum_k W_P(k) P(Z = k|\ell), \tag{20}$$

The LHS term is the sum fitness value for each possible category k, weighted by the probability of this category given a latent value ℓ. Let's use a tangible example: imagine that in a population of bird composed of dispersers and non-dispersers, we are interested in the evolution of dispersal. We measure a fitness value (say life-time reproductive success, LRS) for each category of birds: dispersers have a LRS of 2.5 and non-dispersers have a LRS of 2. Now, if we assume a binomial/probit model, we also know that the probability of "success" (here, say, of being a disperser) given a latent value ℓ is $g^{-1}(\ell)$ so:

$$W_{\text{exp}}(\ell) = 2.5g^{-1}(\ell) + 2\left(1 - g^{-1}(\ell)\right) \tag{21}$$

We will use this function to "translate" selection on the observed data scale (i.e. the *actual* phenotype) to the latent scale (i.e. a hypothetical, but nicely behave phenotype).
To compute the selection gradient, we also need the derivative of the function in Eq. 20, which in our case is:

$$\frac{\mathrm{d}W_{\text{exp}}(\ell)}{\mathrm{d}\ell} = 2.5\frac{\mathrm{d}g^{-1}(\ell)}{\mathrm{d}\ell} - 2\frac{\mathrm{d}g^{-1}(\ell)}{\mathrm{d}\ell} \tag{22}$$

We will soon seen that this derivative is easy to implement in R for this particular case. This derivative is used by `QGpred` to obtain the predicted shift in the latent intercept μ due to selection:

$$\Delta\mu = V_A E\left[\frac{\mathrm{d}W_{\text{exp}}}{\mathrm{d}\ell}\right]\frac{1}{\bar{W}}. \tag{23}$$

Running QGpred To illustrate how to run `QGpred`, we will re-use the example above, with the following estimates from a GLMM binomial/probit analysis:

```
va <- 0.93
vp <- 1.76
mu <- -0.5
```

The inverse of the probit link happens to be the CDF of a Normal distribution, so in R, Eq. 21 is actually quite easy to write:

```
fit <- function(x){2.5 * pnorm(x) + 2 * (1 - pnorm(x))}
```

The derivative of a CDF of a distribution is its PDF, so writing Eq. 22 is quite easy in R:

```
d.fit <- function(x){2.5 * dnorm(x) - 2 * dnorm(x)}
```

Now, we have everything we need to run `QGpred`[13]:

```
QGpred(mu = mu, var.a = va, var.p = vp,
       fit.func = fit, d.fit.func = d.fit)
```

[13] One might be surprised that the model (distribution + link function) assumed is not required as an argument by `QGpred`. In fact, all the information about the model is already accounted for when we wrote `fit` and `d.fit`, nothing else about the distribution is needed.

```
[1] "Computing mean fitness..."
[1] "Computing the latent selection and response..."
  mean.lat.fit   lat.grad    lat.sel   lat.resp
1      2.19086 0.05237713 0.09218375 0.04871073
```

The function yields different information: `mean.lat.fit` is the mean latent fitness ($\bar{W}$ in Eq. 23), `lat.grad` is the latent gradient of selection ($E\left[\frac{\mathrm{d}W_{\mathrm{exp}}}{\mathrm{d}\ell}\right]\frac{1}{\bar{W}}$), `lat.sel` is the latent shift in the selected population and `lat.resp` is the latent response to selection, i.e. the latent shift in the next population.
Obtaining the response to selection on the observed data scale from there is quite straightforward using the function `QGmean` provided by the package[14]:

```
delta.mu <- QGpred(mu = mu, var.a = va, var.p = vp,
                   fit.func = fit, d.fit.func = d.fit)[["lat.resp"]]
QGmean(mu = mu, var = vp,
       link.inv = QGlink.funcs("binom1.probit")[["inv.link"]])
QGmean(mu = mu+delta.mu, var = vp,
       link.inv = QGlink.funcs("binom1.probit")[["inv.link"]])
```

```
0.3817207
0.3929478
```

So, despite quite a strong selection (dispersers have 25% more offspring in their life-time than non-dispersers), we expect the proportion of dispersers to only increase of roughly 3% from this generation to the next.

5 Particular cases

5.1 Including fixed effects

Case study Imagine the phenotypic trait you are studying is the juvenile survival to a common disease. You suspect there is genetic component to it, but at the same time, you know that survival varies at look according to the nutrient intake provided by the parents, which you have a way to measure quite precisely. One way to analyse these data is to include the nutrient intake as a fixed effect in the model, while inferring the additive genetic variance of the survival to the disease. Yet, as stated in section 2.4, including fixed effects in GLMM has strong consequences on the inference of quantitative genetics parameters on the observed data scale. Fortunately, QGglmm has a way to deal with fixed effects by averaging over them: this allows to produce meaningful parameters on the observed data scale.

[14] The `QGmean` fu

Getting the predicted values Using the example above, we ran an animal model in MCMCglmm like this:

```
model <- MCMCglmm(phen ~ nutr, random = ~animal,
                  pedigree = pedigree, data = data,
                  prior = prior, family = "ordinal")
```

where `nutr` is the nutrient intake fixed effect. To account for fixed effects in `QGparams`, we need the *marginal* predicted values of the model. What this means is that predicted values must be computed using only the fixed-effect part of the model, not the random part (e.g. not including the breeding values!). Most implementation of `predict` in R defaults to marginal prediction, and at least `predict.MCMCglmm` does, in our case. What we also need is for predictions to be computed on the latent scale, not on the observed/expected data scale. Here, the default for `predict.MCMCglmm` does not suit us and we need to custom it a bit:

```
yhat <- predict(model, type = "terms")
```

The argument `type = "terms"` tells `predict.MCMCglmm` to compute predictions on the latent scale. This is, of course, just an example: depending on the software you are using, getting the predicted values might be totally different. The important thing is to check they are marginally computed and on the latent scale.

Running QGparams with fixed effects Once these values are obtained, running `QGparams` is pretty straightforward:

```
va <- mean(model[["VCV"]][, "animal"])
vp <- mean(rowSums(model[["VCV"]]))
QGparams(predict = yhat, var.a = va, var.p = vp,
         model = "binom1.probit")
```

```
[1] "Using the closed forms for a Binomial1-probit model."
   mean.obs   var.obs  var.a.obs   h2.obs
1 0.2896591 0.2057567 0.04275686 0.207803
```

Note that `mu` being already accounted for in `predict`, it is not necessary any more[15]. The difference with the output, shouldn't we have used the `predict` argument is quite striking:

```
mu <- mean(model[["Sol"]][, "(Intercept)"])
QGparams(mu = mu, var.a = va, var.p = vp, model = "binom1.probit")
```

```
[1] "Using the closed forms for a Binomial1-probit model."
   mean.obs   var.obs  var.a.obs    h2.obs
1 0.6582622 0.2249531 0.06650048 0.2956193
```

Admittedly, the example here was forged so that these outputs were different, but it is certainly not an unlikely one!

[15] Actually, when using `predict`, the function makes sure that `mu` is not used at all, even it is passed as an argument!

5.2 Obtaining intra-class correlation (ICC) coefficients

What are intra-class correlation coefficients? Intra-class correlation (ICC) measure how much of the data belonging to a given class (e.g. individual, maternal identity, environment, etc.) resemble each other. In other words, it measures how much we can *predict* the look of the data, simply by knowing to which class it belongs. In LMM, formally, ICC are simply the ratio of the variance of the focus component (individuals in the case of repeatability) to the total phenotypic variance. One famous ICC example is the repeatability. It is computed as the ratio of the variance of individual effect (say, V_{I}) to the phenotypic variance :

$$r^2 = \frac{V_{\mathrm{I}}}{V_{\mathrm{P}}}, \tag{24}$$

and quantify how consecutive measurements from the same individual are similar to each other. As yet another example, the heritability can be considered as an ICC measuring how much we can *predict* the phenotype, using e.g. relatedness data.

What about GLMM? As always, the situation is more complex in GLMM. First, computing the variance component on the observed data scale (i.e. for the actual phenotype) is even more complex that computing the additive genetic variance on this scale (de Villemereuil *et al.*, 2016). Second, the link between heritability and ICC is blurred. Narrow-sense heritability on the observed data scale can no longer be considered an ICC. Broad-sense heritability can be computed as an ICC on that level, but it contains some non-additive genetic variance.

Case study Let's say we have multiple individual measurement of a count data set and performed a Poisson/log GLMM using "individual" as a random effect beside the additive genetic one. We obtained the following estimates:

```
mu <- 1
vi <- 0.5        #Individual-effect variance
va <- 0.3        #Additive genetic variance
vp <- 1
```

Now, what we would like to know is, how are measures repeatable on the observed scale? To do so, we will use the function `QGicc`.

Running QGicc Running `QGicc` is very similar to running `QGparams`:

```
QGicc(mu = mu, var.comp = vi + va, var.p = vp, model = "Poisson.log")
```

```
[1] "Using the closed forms for a Poisson-log model."
  mean.obs var.obs var.comp.obs   icc.obs
1 4.481689 38.9943     24.61565 0.6312627
```

Just as `QGparams`, `QGicc` returns the population mean and phenotypic variance that helps to put estimates in context. The parameter `var.comp.obs` is the component variance (here the individual-effect variance) on the observed data scale and `icc.obs` is the ratio of this variance component to the phenotypic variance on the observed data scale.

Some characteristics and other features As said above, computing variance component on the observed data scale is a bit complex. In general, it requires the use of a double integrate, which can sometimes take times. Closed forms are available only for Poisson/log and Negative-Binomial/log models. Binomial/probit models uses a semi-closed form, meaning that a close-form is known for one of the integral, but not the other, so the computation is much faster than we using a double integral, but still slower than when using a fully closed-form. Basically, `QGicc` has all the features `QGparams` has: fixed-effects can be accounted for using the `predict` function and it has a multivariate counter-part `QGmvicc` which yields variance-covariance matrix on the observed data scale. Note that this multivariate function, like `QGmvparams` never uses closed forms. As a consequence, because of the double integration, it is the slowest function of the package[16]!
Finally, note that the "ordinal" model cannot be used with `QGicc` due to its complexity and lack of closed forms for ICC.

5.3 Integrating over a posterior prediction

Why integrate over a posterior distribution? When using a Bayesian implementation of a GLMM (e.g. MCMCglmm), we obtain posterior distributions, rather than point estimates for the parameters on the latent scale. The nice thing about posterior distributions is that they convey a very complex information and yet are quite easy to handle for further transformation of the inference (especially with sampling algorithms such as MCMC). For example, the posterior distribution of heritability is easily computed from a MCMCglmm output as:

```
h2.post <- model[["VCV"]][, "animal"] / rowSums(model[["VCV"]])
```

We can then study directly the shape of `h2.post`, e.g. to determine whether it is different from 0 or not.

Can this be done with QGparams? Absolutely. Recall that the output of `QGparams` is a row of a `data.frame`, which are easily stackable. The only tricky part is to create a `data.frame` to stock our model output, because it's easier to deal with:

[16]Note that when QGglmm switched to cubature, the speed of this function was increased by more than x300, meaning a computation of 25 minutes in the old version now takes... 4 seconds. If you were annoyed by its slugginess, I encourage you to give it another try now!

```
df <- data.frame(mu = as.vector(model[["Sol"]][, "(Intercept)"]),
                 va = as.vector(model[["VCV"]][, "animal"]),
                 vp = rowSums(model[["VCV"]]))
head(df)
```

```
         mu       va       vp
1 0.7329956 1.715276 2.715276
2 0.7444525 1.783855 2.783855
3 0.7700467 1.707488 2.707488
4 0.6952074 1.728199 2.728199
5 0.4953782 2.012740 3.012740
6 0.4433023 1.448746 2.448746
```

Then, we use `apply` to go through this `data.frame` and "stack" our output as one `data.frame` with `do.call("rbind", ...)`:

```
post <- do.call("rbind", apply(df, 1, function(row){
            QGparams(mu = row[["mu"]],
                     var.a = row[["va"]],
                     var.p = row[["vp"]],
                     model = "binom1.probit", verbose = FALSE)
        }))
head(post)
```

```
     mean.obs   var.obs  var.a.obs    h2.obs
va  0.6481320 0.2280569 0.06358546 0.2788140
va1 0.6490326 0.2277893 0.06480902 0.2845130
va2 0.6553937 0.2258528 0.06246510 0.2765744
va3 0.6405957 0.2302328 0.06480583 0.2814795
va4 0.5976603 0.2404625 0.07509439 0.3122915
va5 0.5943345 0.2411010 0.06315448 0.2619420
```

Now, you have the posterior distribution of all the parameters on the observed data scale.

Can this be done with QGmvparams? Absolutely. But it does require a bit more work. The best way to go about it[17] is to create a large data.frame for which each element corresponds to one iteration, and which will contain all latent parameters:

```
df <- cbind(mu1 = model[["Sol"]][, "traitphen1"],
            mu2 = model[["Sol"]][, "traitphen2"],
            model[["VCV"]])
```

Now, all we need to do is construct the matrix "on the fly" before calling `QGmvparams`:

[17] At least from the top of my head, but you can have your own –possibly better– strategy of course!

```
post <- apply(df, 1, function(row) {
  mu <- c(row["mu1"], row["mu2"])
  G <- matrix(c(row["traitphen1:traitphen1.animal"],
                row["traitphen1:traitphen2.animal"],
                row["traitphen1:traitphen2.animal"],
                row["traitphen2:traitphen2.animal"]),
                ncol = 2)
  R <- matrix(c(row["traitphen1:traitphen1.units"],
                row["traitphen1:traitphen2.units"],
                row["traitphen1:traitphen2.units"],
                row["traitphen2:traitphen2.units"]),
                ncol = 2)
  P <- G + R
  QGmvparams(mu = mu, vcv.G = G, vcv.P = P,
       model = c("binom1.probit", "Gaussian"), verbose = FALSE)
})
```

Of course, all names ("phen1", "phen2", etc.) should be changed to fit your actual code, as should your selection of models in QGmvparams.

5.4 The special case of ordinal data

What is an ordinal trait? An ordinal trait is a categorical trait for which the categories have a meaningful order[18]. Think about, for example, a morphological trait categorised as "small", "intermediate" and "large" or a disease which could manifest a "asymptomatic", "unwell" and "severe". It is pretty clear that these categories are ordered and what the order is (whether you begin by small or large doesn't really matter).

How are these traits studied? The most common mod el to study those traits in quantitative genetics is called the *multiple threshold model*. In this model, a Gaussian variable (the "liability") is cut into separate parts by cut-points (or thresholds) corresponding to each categories (see Fig. 5), e.g. defining their relative proportions. Because of unidentifiability issues, one of the cut-points (or the intercept μ) has to be arbitrarily set. Thus, most often, the first cut-point is set to be 0 (see Fig. 5).

The GLMM specification In the GLMM realm, this model can be translated as a multionial/probit model where the probit link is a bit peculiar: it is "sliced" according to the different cut-points (hereafter γ_i), corresponding to the multiple thresholds. As a consequence, the link function outputs a vector containing the relative probability for each category, all of which sum

[18]Though binary traits are a special case of ordinal trait, they are excluded here for two reasons. First, they are a very peculiar, degenerate kind of ordinal trait. Second, the core function of QGglmm can deal with them as binomial traits much efficiently.

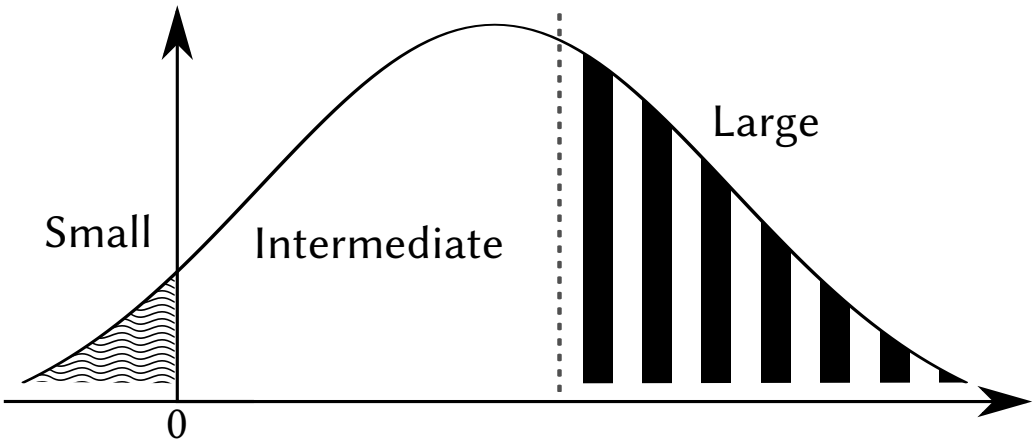

Figure 5: A schematic representation of the multiple threshold model using the example of the "small", "intermediate" and "large" categories. The area under the curve shows the relative proportions of each category: here "small" would be the rarest one.

up to 1. This vector is then used to sample a (unique) value in a multinomial distribution:

$$\boldsymbol{\ell} = \mu + \mathbf{Xb} + \mathbf{Z_a a} + \mathbf{Z_1 u_1} + ... + \mathbf{Z}_k \mathbf{u_k} + \mathbf{o} \tag{25a}$$

$$\boldsymbol{p}_i(\boldsymbol{\ell}) = \Phi(\gamma_i - \boldsymbol{\ell}) - \Phi(\gamma_{i-1} - \boldsymbol{\ell}) \tag{25b}$$

$$\boldsymbol{Z} \sim \text{Multinom}(\boldsymbol{p}_1, ..., \boldsymbol{p}_K) \tag{25c}$$

where Φ is a standard Normal CDF (inverse of the probit link) and K the number of categories. Note that the equations above suppose that there are two "extreme" cut-points $-\infty$ and ∞. Something important to note here is that $\boldsymbol{Z}$ is a matrix of K rows and as many columns as there are individuals. Each column has 1 for the expressed phenotypic category and 0 elsewhere. As a consequence, despite the latent being univariate, the observed data scale is actually multivariate and of dimension K!

Case study Let's assume you have a phenotypic trait with 3 categories (small, intermediate, large). You want to study the quantitative genetics behind this trait and run a multinomial GLMM for this[19]. You obtain the following parameters:

```
mu <- 1.5
va <- 0.5
vp <- 1
cp <- c(0, 2) # Estimated cut-points
```

An illustrative simulation To fully understand the model and what these parameters stand for, let's simulate a few data point according to them. First, let's fix a few things:

[19]Note that the package MCMCglmm has both the "multinomial" and "ordinal" models. The former one actually uses a logit link, which is not implemented in QGglmm (yet?), while the latter one corresponds to the model described here.

```
# Number of simulated individuals
N <- 10
# All cut-points including the infinite extremes
g <- c(-Inf, cp, Inf)
```

Now, we can proceed. The latent values are pretty straightforward, as always:

```
l <- mu + rnorm(N, 0, sqrt(va)) + rnorm(N, 0, sqrt(vp-va))
l
```

```
[1] 1.3991485 1.4866290 1.7736644 0.9845936 1.3013626 1.5379076 3.4074896
[8] 0.7511216 0.4951887 3.3479212
```

As we can see, `l` is just a vector containing 10 values (one for each simulated individual). Now, the complicated part is to use the "sliced" probit:

```
p = matrix(0, ncol = N, nrow = 3)
for (i in 1:3) {
  p[i, ] =  pnorm(g[i + 1] - l) - pnorm(g[i] - l)
}
p
```

```
      [, 1]       [, 2]       [, 3]      [, 4]       [, 5]      [, 6]
[1, ] 0.08088423 0.06855641 0.03805934 0.1624119 0.09656718 0.0620356
[2, ] 0.64514632 0.62759767 0.55147046 0.6826321 0.66104348 0.6159569
[3, ] 0.27396945 0.30384592 0.41047020 0.1549561 0.24238933 0.3220075
      [, 7]      [, 8]      [, 9]        [, 10]
[1, ] 0.000327817 0.2262897 0.3102334 0.0004071008
[2, ] 0.079313308 0.6678555 0.6235802 0.0884347631
[3, ] 0.920358875 0.1058548 0.0661863 0.9111581361
```

We can see that for each of the ten individuals, we obtain the probabilities to belong to each of the category (first row is small, etc.). We can also see the relationship between the latent values `l` and the probabilities in `p`. For example, the 7th and 10th values in `l` are quite large:

```
l[c(7, 10)]
```

```
[1] 3.407490 3.347921
```

leading to very strong probabilities to belong to the last category:

```
p[, c(7, 10)]
```

```
             [, 1]         [, 2]
[1, ] 0.000327817 0.0004071008
[2, ] 0.079313308 0.0884347631
[3, ] 0.920358875 0.9111581361
```

Finally, we can use the multinomial distribution to sample the actual phenotypic category of the individuals given these probabilities:

```
z <- apply(p, 2, function(vec) {
  vec <- as.vector(vec)
  rmultinom(1, size = 1, prob = vec)
})
z
```

```
     [, 1] [, 2] [, 3] [, 4] [, 5] [, 6] [, 7] [, 8] [, 9] [, 10]
[1, ]    0    0    0    0    0    0    0    0    0     0
[2, ]    0    0    0    1    1    0    0    1    1     0
[3, ]    1    1    1    0    0    1    1    0    0     1
```

Individuals with 1 on row 1 (here none) have a phenotype "small", those with 1 on row 2 have a phenotype "intermediate" and those with 1 on row 3 have a phenotype "large".

Running QGparams for ordinal traits Running `QGparams` for ordinal is actually quite easy, yet the output can be surprising:

```
out <- QGparams(mu = mu, var.a = va, var.p = vp,
                model = "ordinal", cut.points = c(-Inf, cp, Inf))
out
```

```
$mean.obs
[1] 0.1444222 0.4937410 0.3618368

$vcv.P.obs
            [, 1]        [, 2]        [, 3]
[1, ]  0.12356442 -0.07130715 -0.05225726
[2, ] -0.07130715  0.24996083 -0.17865367
[3, ] -0.05225726 -0.17865367  0.23091093

$vcv.G.obs
             [, 1]         [, 2]        [, 3]
[1, ]  0.012917511  0.008379864 -0.02129738
[2, ]  0.008379864  0.005436196 -0.01381606
[3, ] -0.021297376 -0.013816061  0.03511344

$h2.obs
[1] 0.10454071 0.02174819 0.15206485
```

Despite being a univariate analysis, the output is a multivariate one. It is a `list`, not a `data.frame` and contains variance-covariance matrices and not variances. Also, it contains one heritability estimate for each category. Importantly, the G-matrix on the observed data scale is a $K \times K$ matrix, but has only one axis of variation:

```
# Only 1 non-null eigen value
eigen(cov2cor(out[["vcv.G.obs"]]))[["values"]]
```

```
[1] 3.000000e+00 8.881784e-16 0.000000e+00
```

References

de Villemereuil P., Schielzeth H., Nakagawa S. & Morrissey M. (2016). General methods for evolutionary quantitative genetic inference from generalised mixed models. *Genetics* p. genetics.115.186536. doi:10.1534/genetics.115.186536. Cited page(s): 2, 4, 7, 9, 10, 11, 20, 24

Fisher R.A. (1918). The correlation between relatives on the supposition of Mendelian inheritance. *Transactions of the Royal Society of Edinburgh* **52**: 399–433. Cited page(s): 3, 7

Henderson C.R. (1950). Estimation of genetic parameters. *Annals of Mathematical Statistics* **21**: 309–310. Cited page(s): 2

Henderson C.R. (1976). A simple method for computing the inverse of a numerator relationship matrix used in prediction of breeding values. *Biometrics* **32**: 69–83, URL: http://www.jstor.org/stable/2529339. Cited page(s): 2

Kruuk L.E.B. (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. doi:10.1098/rstb.2003.1437. Cited page(s): 2

Nakagawa S. & Schielzeth H. (2010). Repeatability for Gaussian and non Gaussian data: a practical guide for biologists. *Biological Reviews* **85**: 935–956. doi:10.1111/j.1469-185X.2010.00141.x. Cited page(s): 4, 8