

Advanced biostatistics and Open science

Lab manual for BIO8940

Julien Martin

2024-04-25

Table of contents

Note	1
I. Open Science	3
1. Introduction to open Science	5
1.1. Why do we need it?	5
1.2. Lecture	5
1.3. What it is?	5
1.4. Reproducible code and analysis	5
2. Introduction to Rmarkdown	7
2.1. Lecture	7
2.2. Practical	7
3. Introduction to github with R	13
3.1. Lecture	13
3.2. Practical	13
II. Statistics	17
4. Generalized linear model, glm	19
4.1. Lecture	19
4.2. Practical	20
5. Introduction to linear mixed models	33
5.1. Lecture	33
5.2. Practical	37
6. Introduction to GLMM	59
6.1. Lecture	59
6.2. Practical	59
7. Introduction to Bayesian Inference	89
7.1. Lecture	89
7.2. Practical	95
8. Multivariate mixed models	115
8.1. Lecture	115
8.2. Practical	115
9. Random regression and character state approaches	137
9.1. Lecture	137

Table of contents

9.2. Practical	137
10. Beyond $P < 0.05$	163
References	165
R packages	165
Appendices	177
A. R	177

Note

! Important

Work in progress. New chapters are going to appear regularly meaning that if you download the pdf it might be incomplete by the time we do the practical in class.

if you see a dragon in a section, it means it is under development



Figure 1.: Dream pet dragon

Licence

The document is available following the license License Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International.



Figure 2.: License Creative Commons

Part I.

Open Science

1. Introduction to open Science

1.1. Why do we need it?

1.2. Lecture



Figure 1.1.: Dream pet dragon

1.3. What it is?

1.4. Reproducible code and analysis

2. Introduction to Rmarkdown

2.1. Lecture



Figure 2.1.: Dream pet dragon

2.2. Practical

We will create a new Rmarkdown document and edit it using basic R and Rmarkdown functions.

2.2.1. Context

We will use the awesome palmerpenguins dataset 🐧 to explore and visualize data.

These data have been collected and shared by Dr. Kristen Gorman and Palmer Station, Antarctica LTER.

The package was built by Drs Allison Horst and Alison Hill, check out the official website.

The package palmerpenguins has two datasets:

- penguins_raw has the raw data of penguins observations (see ?penguins_raw for more info)
- penguins is a simplified version of the raw data (see ?penguins for more info)


For this exercise, we're gonna use the penguins dataset.

```
library(palmerpenguins)
head(penguins)
```

```
# A tibble: 6 x 8
  species island   bill_length_mm bill_depth_mm flipper_length_mm body_mass_g
  <fct>   <fct>         <dbl>         <dbl>         <int>         <int>
1 Adelie  Torgersen       39.1           18.7           181           3750
2 Adelie  Torgersen       39.5           17.4           186           3800
3 Adelie  Torgersen       40.3           18             195           3250
4 Adelie  Torgersen       NA             NA             NA            NA
5 Adelie  Torgersen       36.7           19.3           193           3450
6 Adelie  Torgersen       39.3           20.6           190           3650
# i 2 more variables: sex <fct>, year <int>
```

2.2.2. Questions

1) Install the package palmerpenguins.

 Solution

```
install.packages("palmerpenguins")
```

2)

- Create a new R Markdown document, name it and save it.
- Delete everything after line 12.
- Add a new section title, simple text and text in bold font.
- Compile (“Knit”).


3)

- Add a chunk in which you load the palmerpenguins. The corresponding line of code should be hidden in the output.
- Load also the tidyverse suite of packages. Modify the defaults to suppress all messages.

 Solution

```
```${r, echo = FALSE, message = FALSE}
library(palmerpenguins)
library(tidyverse)
```
```

4) Add another chunk in which you build a table with the 10 first rows of the dataset.

 Solution

```

```{r}
penguins %>%
 slice(1:10) %>%
 knitr::kable()
```

```

5) In a new section, display how many individuals, penguins species and islands we have in the dataset. This info should appear directly in the text, you need to use inline code 😊. Calculate the mean of the (numeric) traits measured on the penguins.

💡 Solution

```
## Numerical exploration
```

There are ``r nrow(penguins)`` penguins in the dataset,
and ``r length(unique(penguins$species))`` different species.
The data were collected in ``r length(unique(penguins$island))``
islands of the Palmer archipelago in Antarctica.

The mean of all traits that were measured on the penguins are:

```

```{r echo = FALSE}
penguins %>%
 group_by(species) %>%
 summarize(across(where(is.numeric), mean, na.rm = TRUE))
```

```

6) In another section, entitled 'Graphical exploration', build a figure with 3 superimposed histograms, each one corresponding to the body mass of a species.

💡 Solution

Graphical exploration

A histogram of body mass per species:

```
```{r, fig.cap = "Distribution of body mass by species of penguins"}
ggplot(data = penguins) +
 aes(x = body_mass_g) +
 geom_histogram(aes(fill = species),
 alpha = 0.5,
 position = "identity") +
 scale_fill_manual(values = c("darkorange", "purple", "cyan4")) +
 theme_minimal() +
 labs(x = "Body mass (g)",
 y = "Frequency",
 title = "Penguin body mass")
```
```

7) In another section, entitled *Linear regression*, fit a model of bill length as a function of body size (flipper length), body mass and sex. Obtain the output and graphically evaluate the assumptions of the model. As reminder here is how you fit a linear regression.

```
```{r}
model <- lm(Y ~ X1 + X2, data = data)
summary(model)
plot(model)
```
```

💡 Solution

Linear regression

And here is a nice model with graphical output

```
```{r, fig.cap = "Checking assumptions of the model"}
m1 <- lm(bill_length_mm ~ flipper_length_mm + body_mass_g + sex, data = penguins)
summary(m1)
par(mfrow= c(2,2))
plot(m1)
```
```

8) Add references manually or using `citr` in RStudio.

1. Pick a recent publication from the researcher who shared the data, Dr Kristen Gorman. Import this publication in your favorite references manager (we use Zotero, no hard feeling), and create a bibtex reference that you will add to to the file `mabiblio.bib`.
2. Add bibliography: `mabiblio.bib` at the beginning of your R Markdown document (YAML).

3. Cite the reference in the text using either typing the reference manually or using `citr`. To use `citr`, install it first; if everything goes well, you should see it in the pull-down menu Addins 🍌. Then simply use Insert citations in the pull-down menu Addins.
4. Compile.

9) Change the default citation format (Chicago style) into the The American Naturalist format. It can be found here <https://www.zotero.org/styles>. To do so, add `csl: the-american-naturalist.csl` in the YAML.

10) Build your report in html, pdf and docx format. 🎉

Example of output

You can see an example of the Rmarkdown source file and pdf output

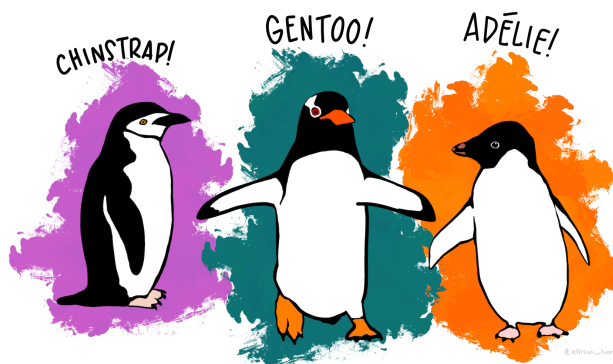


Figure 2.2.: Happy coding

3. Introduction to github with R

3.1. Lecture



Figure 3.1.: Dream pet dragon

3.2. Practical

3.2.1. Context

We will configure Rstudio to work with our github account, then create a new project and start using github. To have some data I suggest to use the awesome palmerpenguins dataset 🐧.

3.2.2. Information of the data

These data have been collected and shared by Dr. Kristen Gorman and Palmer Station, Antarctica LTER.

The package was built by Drs Allison Horst and Alison Hill, check out the official website.

The package palmerpenguins has two datasets.

```
library(palmerpenguins)
```

The dataset penguins is a simplified version of the raw data; see ?penguins for more info:

```
head(penguins)
```

```
# A tibble: 6 x 8
  species island   bill_length_mm bill_depth_mm flipper_length_mm body_mass_g
  <fct>   <fct>         <dbl>         <dbl>         <int>         <int>
1 Adelie Torgersen     39.1           18.7           181           3750
2 Adelie Torgersen     39.5           17.4           186           3800
3 Adelie Torgersen     40.3            18            195           3250
4 Adelie Torgersen     NA              NA              NA             NA
5 Adelie Torgersen     36.7           19.3           193           3450
6 Adelie Torgersen     39.3           20.6           190           3650
# i 2 more variables: sex <fct>, year <int>
```

The other dataset `penguins_raw` has the raw data; see `?penguins_raw` for more info:

```
head(penguins_raw)
```

```
# A tibble: 6 x 17
  studyName `Sample Number` Species      Region Island Stage `Individual ID`
  <chr>          <dbl> <chr>         <chr> <chr> <chr> <chr>
1 PAL0708           1 Adelie Penguin ~ Anvers Torge~ Adul~ N1A1
2 PAL0708           2 Adelie Penguin ~ Anvers Torge~ Adul~ N1A2
3 PAL0708           3 Adelie Penguin ~ Anvers Torge~ Adul~ N2A1
4 PAL0708           4 Adelie Penguin ~ Anvers Torge~ Adul~ N2A2
5 PAL0708           5 Adelie Penguin ~ Anvers Torge~ Adul~ N3A1
6 PAL0708           6 Adelie Penguin ~ Anvers Torge~ Adul~ N3A2
# i 10 more variables: `Clutch Completion` <chr>, `Date Egg` <date>,
# `Culmen Length (mm)` <dbl>, `Culmen Depth (mm)` <dbl>,
# `Flipper Length (mm)` <dbl>, `Body Mass (g)` <dbl>, Sex <chr>,
# `Delta 15 N (o/oo)` <dbl>, `Delta 13 C (o/oo)` <dbl>, Comments <chr>
```

For this exercise, we're gonna use the `penguins` dataset.

3.2.3. Questions

- 1) Create a github account if not done yet.
- 2) Configure Rstudio with your github account using the `usethis` package.
- 3) Create and Store your GITHUB Personal Authorisation Token
- 4) Create a new R Markdown project, initialize it for git, and create a new git repository
- 5) Create a new Rmarkdown document, in your project. Then save the file and stage it.
- 6) Create a new commit including the new file and push it to github (Check on github that it works).

7) Edit the file. Delete everything after line 12. Add a new section title, simple text and text in bold font. Then knit and compile.

8) Make a new commit (with a meaningful message), and push to github.

9) Create a new branch, and add a new section to the rmarkdown file in this branch. Whatever you want. I would suggest a graph of the data.

10) Create a commit and push it to the branch.

11) On github, create a pull request to merge the 2 different branches.

12) Check and accept the pull request to merge the 2 branches.

You have successfully used all the essential tools of git 🎉. You are really to explore 🧑🏻 and discover its power 💪



Figure 3.2.: Happy git(hub)-ing

3.2.4. Solution

2)

```
usethis::git_sitrep()
usethis::use_git_config(
  user.name = "your_username",
  user.email = "your_email@address.com"
)
```

3)

```
usethis::create_github_token()
gitcreds::gitcreds_set()
```

4)

```
#create R project
usethis::use_git()

#restart R
usethis::use_github()
usethis::git_vaccinate()
```

Part II.
Statistics

4. Generalized linear model, glm

4.1. Lecture



Figure 4.1.: Dream pet dragon

```
m1 <- glm(fish ~ french_captain, data = dads_joke, family = poisson)
```

4.1.1. Distributions

4.1.1.1. Continuous linear

- Gaussian

4.1.1.2. Count data

- poisson
- negative binomial
- quasi-poisson
- generalized poisson
- conway-maxwell poisson

4.1.1.3. censored distribution

4.1.1.4. zero-inflated / hurdle distribution

- zero-inflated/zero-truncated poisson
- censored poisson

4.1.1.5. zero-truncated distribution

4.1.1.6. zero-one-inflated distribution

see https://cran.r-project.org/web/packages/brms/vignettes/brms_families.html see also MCMCglmm coursenotes for help on description and to add some plots about those distribution

4.2. Practical

Warning

This section need to be severely updated

4.2.1. Logistic regression

```
library(tidyverse)
library(DHARMA)
```

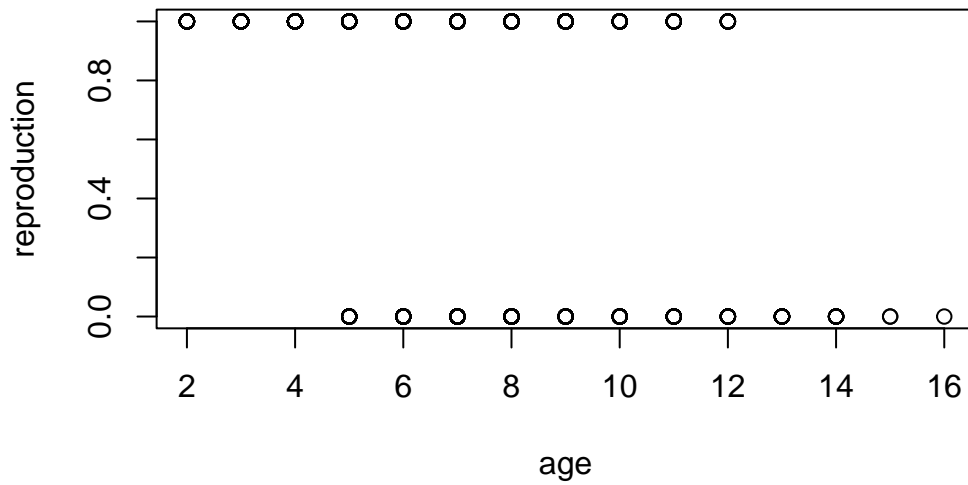
This is DHARMA 0.4.6. For overview type '?DHARMA'. For recent changes, type `news(package = 'DHARMA')`

```
library(performance)

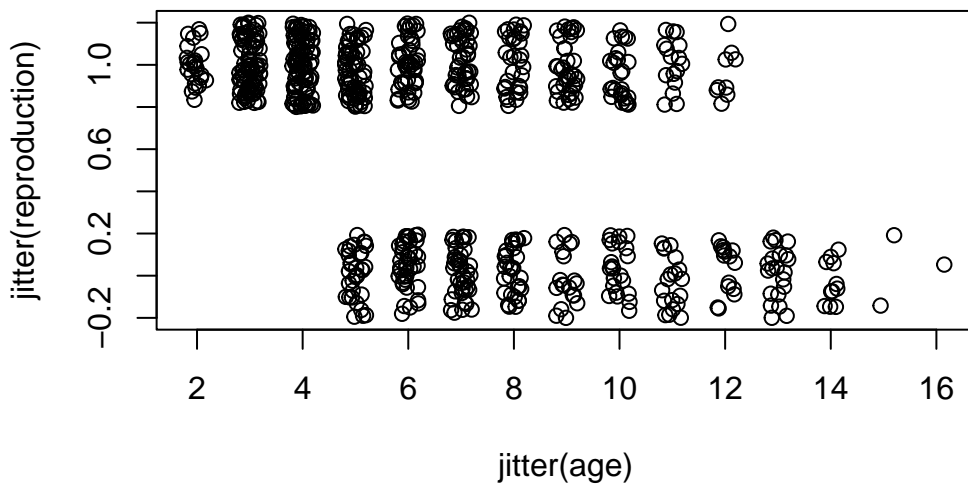
mouflon <- read.csv("data/mouflon.csv")
mouflonc <- mouflon[order(mouflon$age),]

mouflonc$reproduction <- ifelse(mouflonc$age < 13, mouflonc$reproduction, 0)
mouflonc$reproduction <- ifelse(mouflonc$age > 4, mouflonc$reproduction, 1)

plot(reproduction ~ age, mouflonc)
```

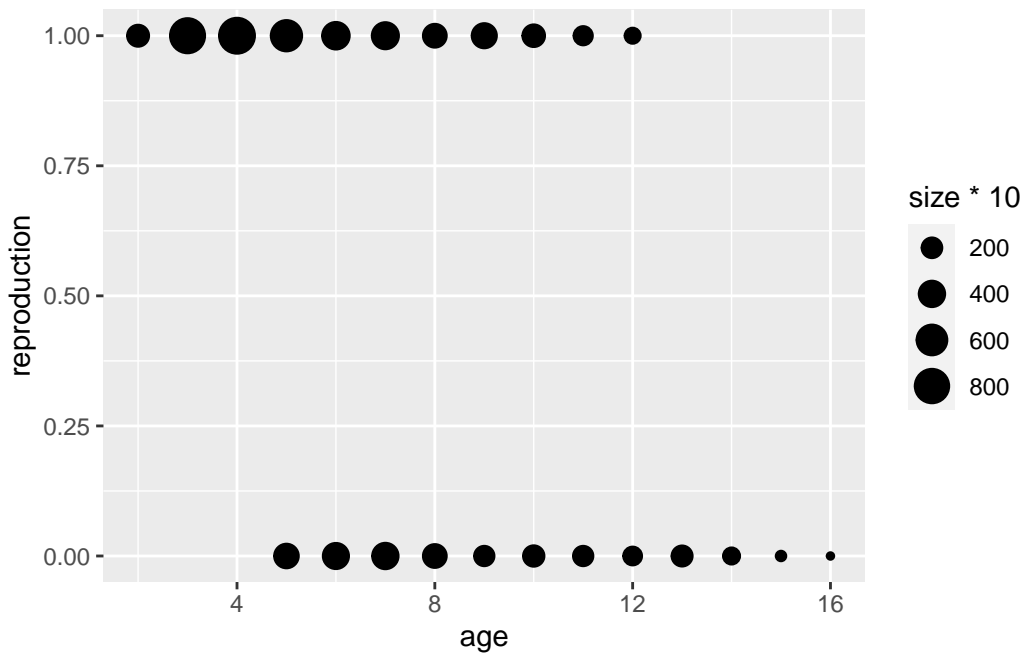



```
plot(jitter(reproduction) ~ jitter(age), mouflonc)
```



```
bubble <- data.frame(age = rep(2:16, 2),
                    reproduction = rep(0:1, each = 15),
                    size = c(table(mouflonc$age, mouflonc$reproduction)))
bubble$size <- ifelse(bubble$size == 0, NA, bubble$size)
ggplot(data = bubble, aes(x = age, y = reproduction)) +
  geom_point(aes(size = size*10))
```

Warning: Removed 7 rows containing missing values (``geom_point()``).



```
m1 <- glm(reproduction ~ age,
  data = mouflonc,
  family = binomial)
summary(m1)
```

Call:

```
glm(formula = reproduction ~ age, family = binomial, data = mouflonc)
```

Coefficients:

| | Estimate | Std. Error | z value | Pr(> z) |
|-------------|----------|------------|---------|------------|
| (Intercept) | 3.19921 | 0.25417 | 12.59 | <2e-16 *** |
| age | -0.36685 | 0.03287 | -11.16 | <2e-16 *** |

Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for binomial family taken to be 1)

Null deviance: 928.86 on 715 degrees of freedom

Residual deviance: 767.51 on 714 degrees of freedom

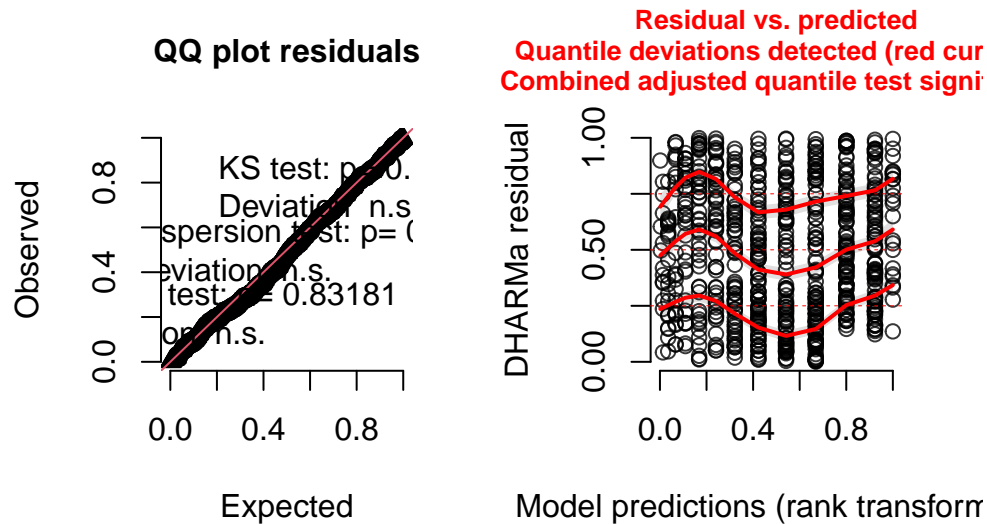
(4 observations deleted due to missingness)

AIC: 771.51

Number of Fisher Scoring iterations: 4

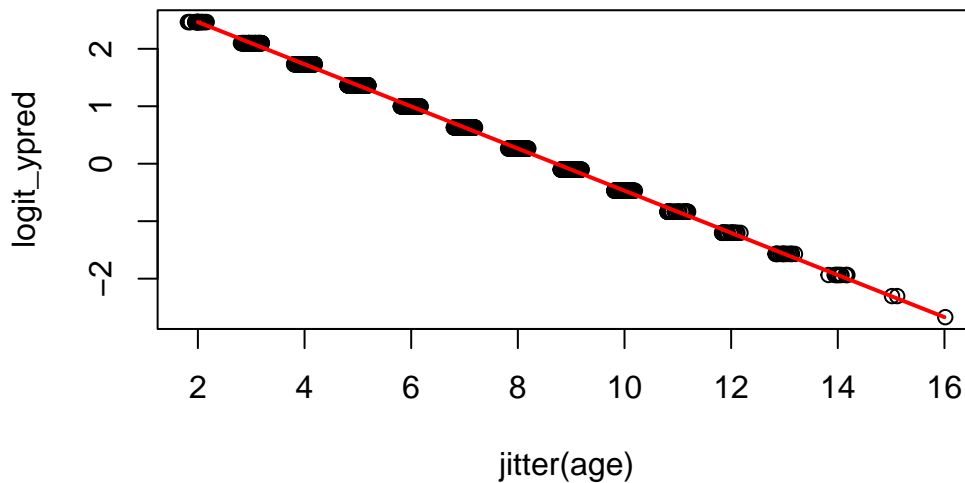
```
simulationOutput <- simulateResiduals(m1)
plot(simulationOutput)
```

DHARMA residual



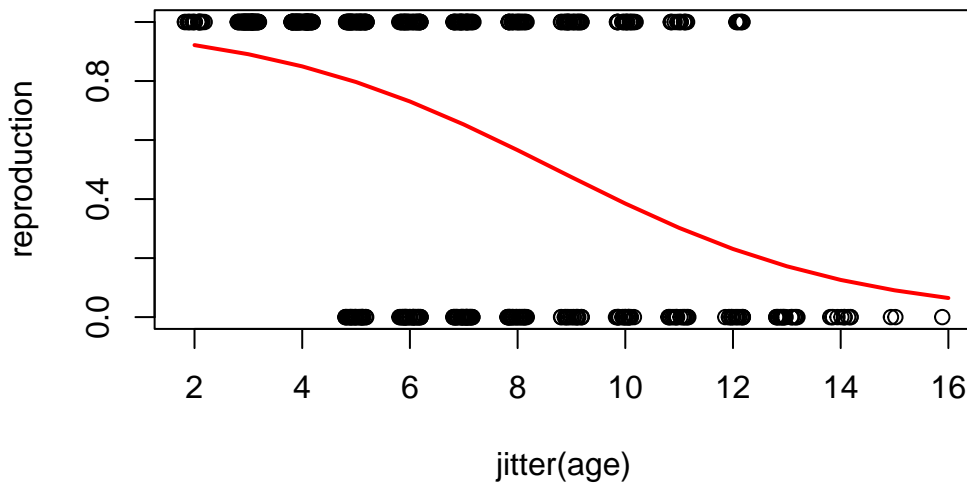
plotting the model prediction on the link (latent) scale

```
mouflonc$logit_ypred <- 3.19921 - 0.36685 * mouflonc$age
plot(logit_ypred ~ jitter(age), mouflonc)
points(mouflonc$age, mouflonc$logit_ypred, col="red", type = "l", lwd = 2)
```



plotting on the observed scale

```
mouflonc$ypred <- exp(mouflonc$logit_ypred) / (1 + exp(mouflonc$logit_ypred)) # inverse of logit
plot(reproduction ~ jitter(age), mouflonc)
points(mouflonc$age, mouflonc$ypred, col="red", type = "l", lwd = 2)
```



Enfin, pour se simplifier la vie, il est aussi possible de récupérer les valeurs prédites de y directement

```
plot(x,y)
myreg <- glm(y~x, family=binomial(link=logit))
ypredit <- myreg$fitted
o=order(x)
points(x[o],ypredit[o], col="red", type="l", lwd=2)
```

```
m2 <- glm(reproduction ~ age + mass_sept + as.factor(sex_lamb) + mass_gain + density + temp,
          data = mouflon,
          family = binomial)
summary(m2)
```

Call:

```
glm(formula = reproduction ~ age + mass_sept + as.factor(sex_lamb) +
     mass_gain + density + temp, family = binomial, data = mouflon)
```

Coefficients:

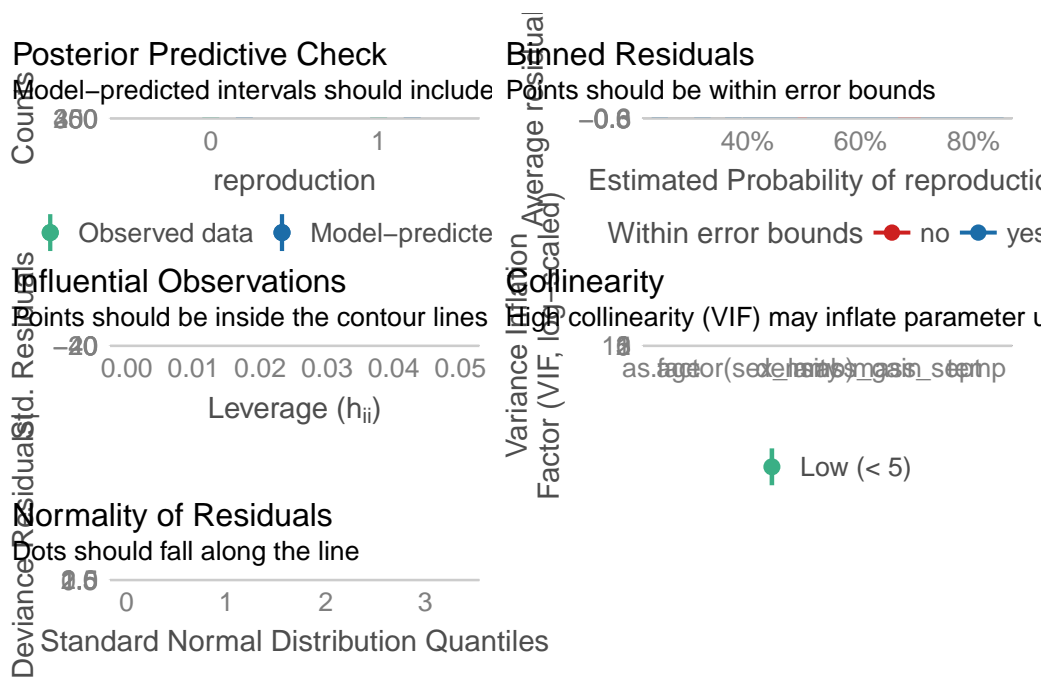
| | Estimate | Std. Error | z value | Pr(> z) |
|----------------------|-----------|------------|---------|--------------|
| (Intercept) | 1.622007 | 1.943242 | 0.835 | 0.403892 |
| age | -0.148567 | 0.033597 | -4.422 | 9.78e-06 *** |
| mass_sept | 0.029878 | 0.016815 | 1.777 | 0.075590 . |
| as.factor(sex_lamb)1 | -0.428169 | 0.166156 | -2.577 | 0.009969 ** |
| mass_gain | -0.094828 | 0.026516 | -3.576 | 0.000348 *** |
| density | -0.018132 | 0.003518 | -5.154 | 2.55e-07 *** |
| temp | 0.037244 | 0.138712 | 0.269 | 0.788313 |

Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for binomial family taken to be 1)

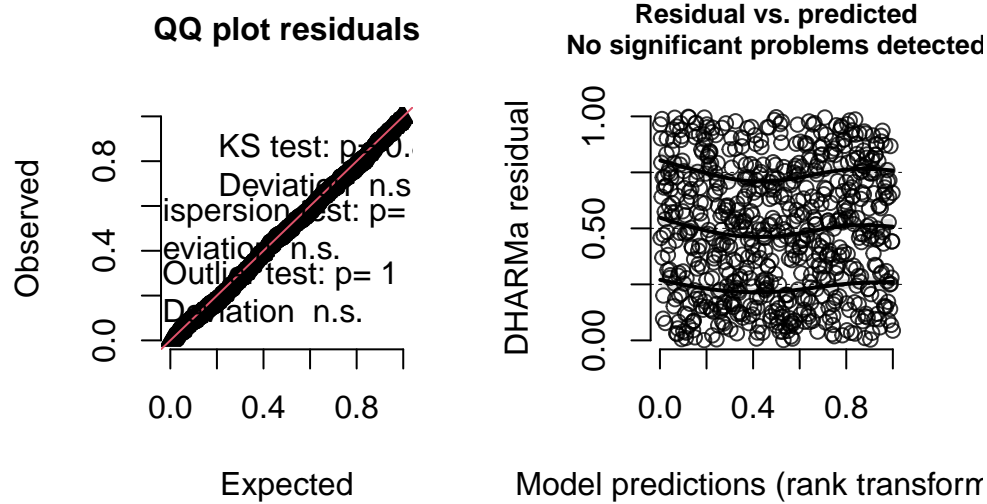
Null deviance: 916.06 on 674 degrees of freedom
 Residual deviance: 845.82 on 668 degrees of freedom
 (45 observations deleted due to missingness)
 AIC: 859.82

Number of Fisher Scoring iterations: 4

`check_model(m2)`

```
simulationOutput <- simulateResiduals(m2)
plot(simulationOutput)
```

DHARMA residual



4.2.1.1. previous offspring sex effect

```

pred.data <- data.frame(
  age = mean(mouflon$age),
  mass_sept = mean(mouflon$mass_sept),
  sex_lamb = c(0,1),
  mass_gain = mean(mouflon$mass_gain),
  density = mean(mouflon$density),
  temp = mean(mouflon$temp, na.rm =TRUE))

predict(m2, newdata = pred.data)

```

1 2
0.6225895 0.1944205

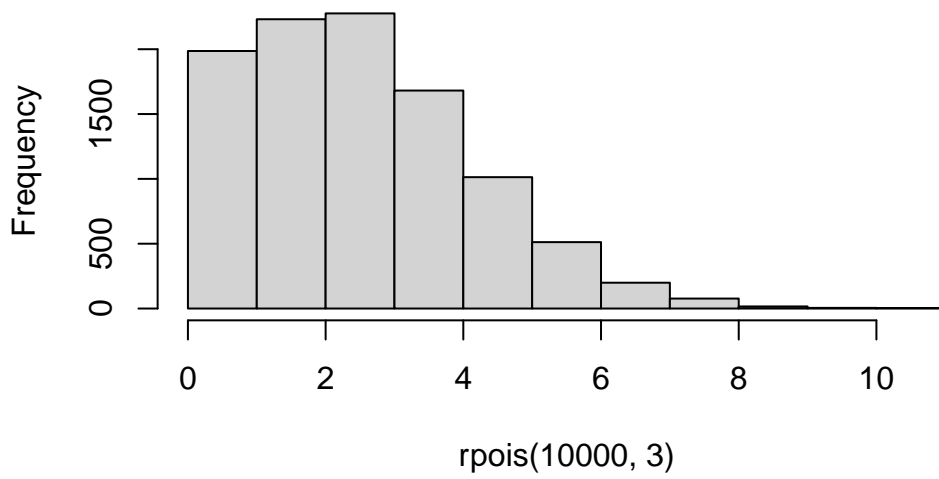
4.2.2. Poisson regression

data on galapagos islands species richness model of total number of species model of proportion of native model of density of species

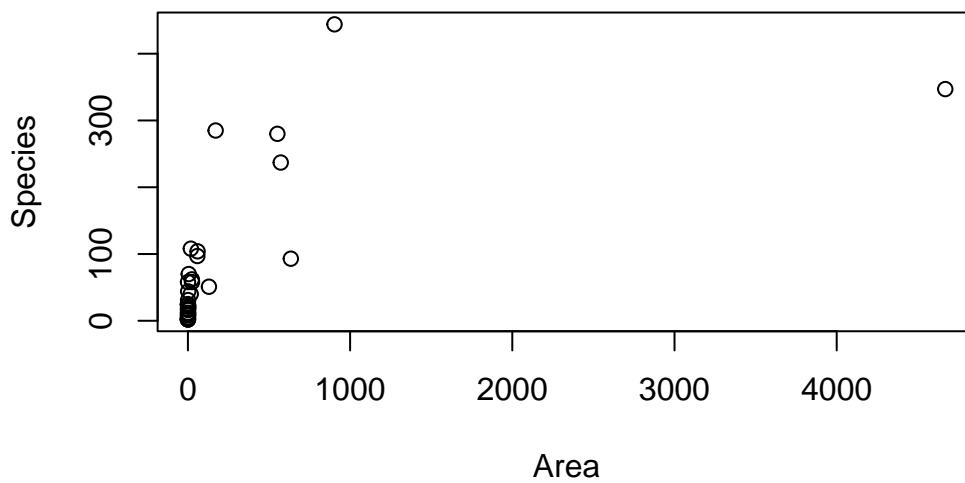
Fit 3 models - model of total number of species - model of proportion of endemics to total - model of species density

```
hist(rpois(10000,3))
```

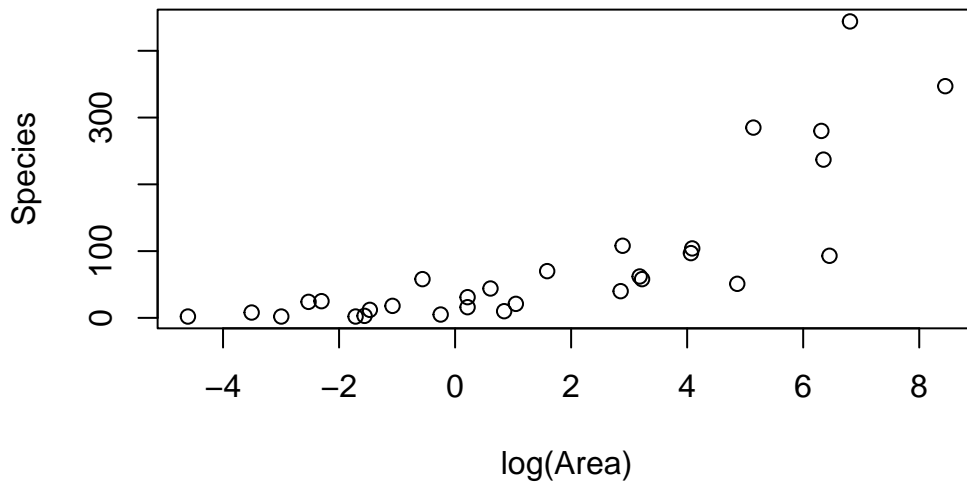
Histogram of rpois(10000, 3)



```
#
gala <- read.delim2("data/gala.txt")
plot(Species ~ Area, gala)
```

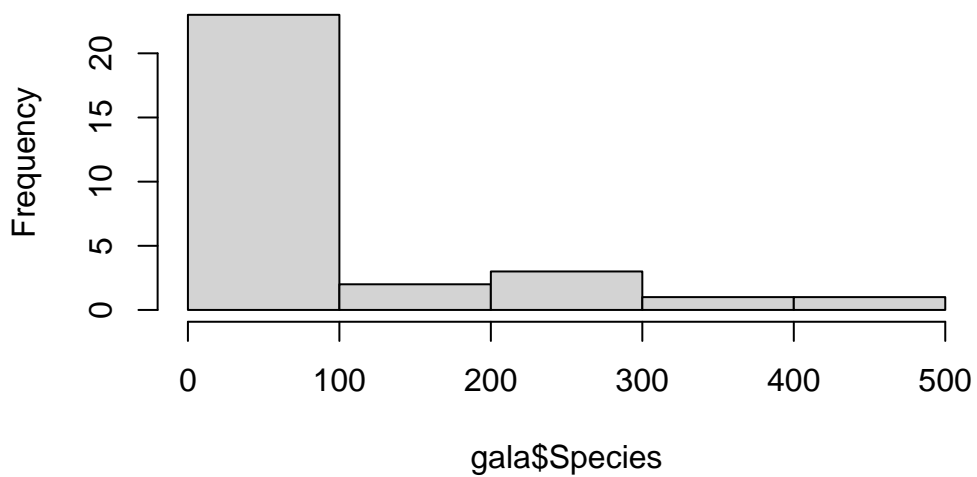


```
plot(Species ~ log(Area), gala)
```



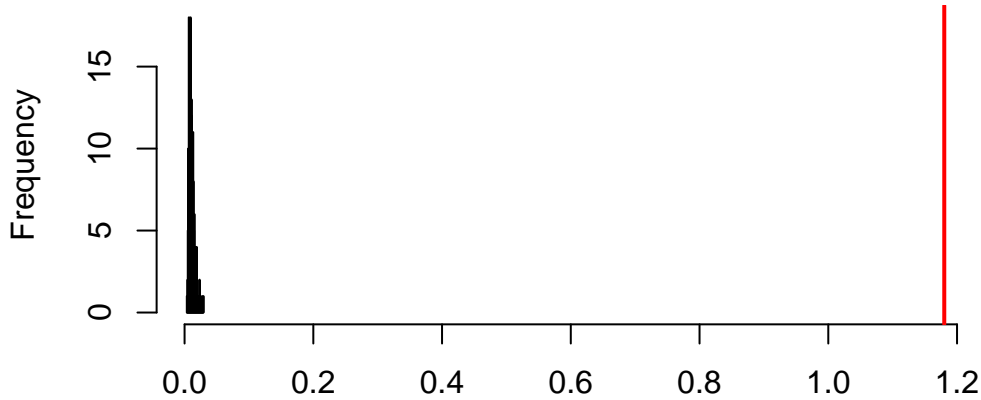
```
hist(gala$Species)
```

Histogram of gala\$Species



```
modpl <- glm(Species ~ Area + Elevation + Nearest, family=poisson, gala)
res <- simulateResiduals(modpl)
testDispersion(res)
```


DHARMA nonparametric dispersion test via sd of residuals fitted vs. simulated



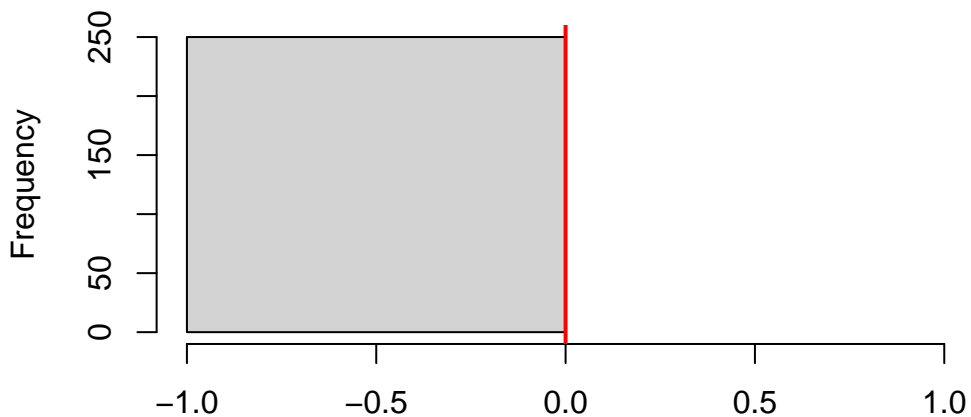
Simulated values, red line = fitted model. p-value (two.sided) = 0

DHARMA nonparametric dispersion test via sd of residuals fitted vs. simulated

```
data: simulationOutput
dispersion = 110.32, p-value < 2.2e-16
alternative hypothesis: two.sided
```

```
testZeroInflation(res)
```

DHARMA zero-inflation test via comparison to expected zeros with simulation under H0 = fitted model



Simulated values, red line = fitted model. p-value (two.sided) = 1

DHARMA zero-inflation test via comparison to expected zeros with simulation under H_0 = fitted model

```
data: simulationOutput
ratioObsSim = NaN, p-value = 1
alternative hypothesis: two.sided
```

```
mean(gala$Species)
```

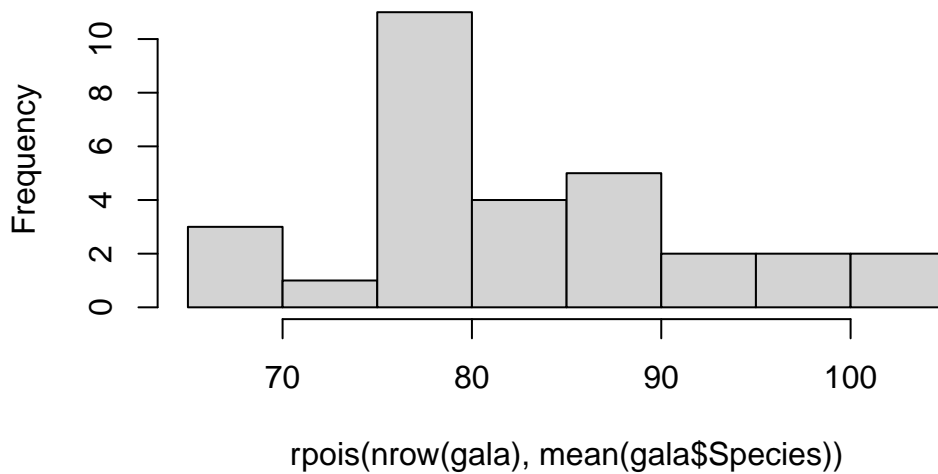
```
[1] 85.23333
```

```
var(gala$Species)
```

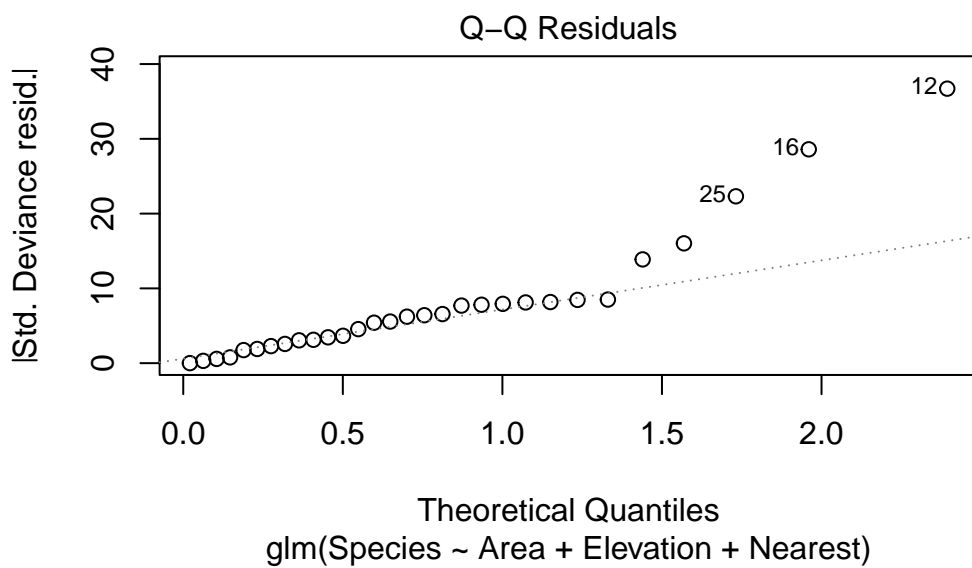
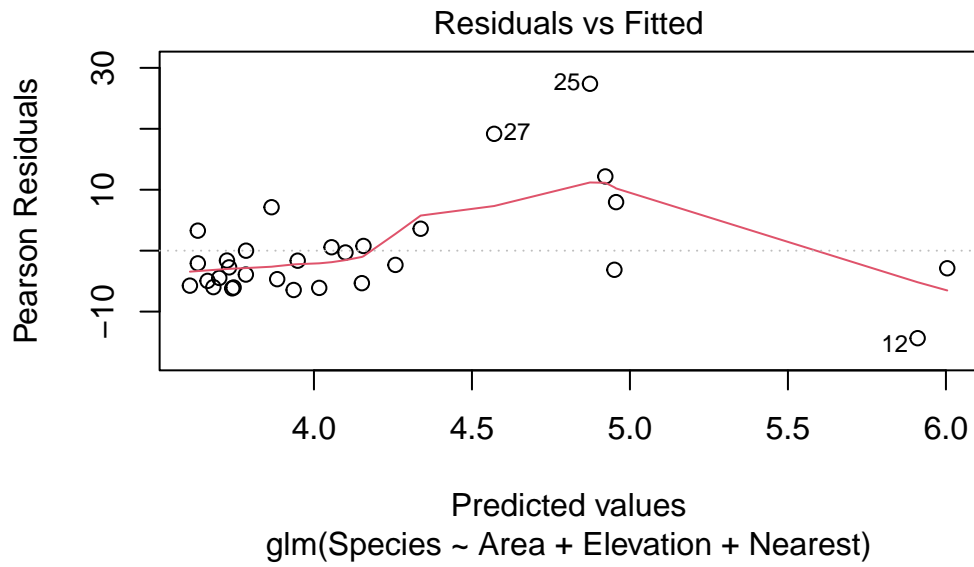
```
[1] 13140.74
```

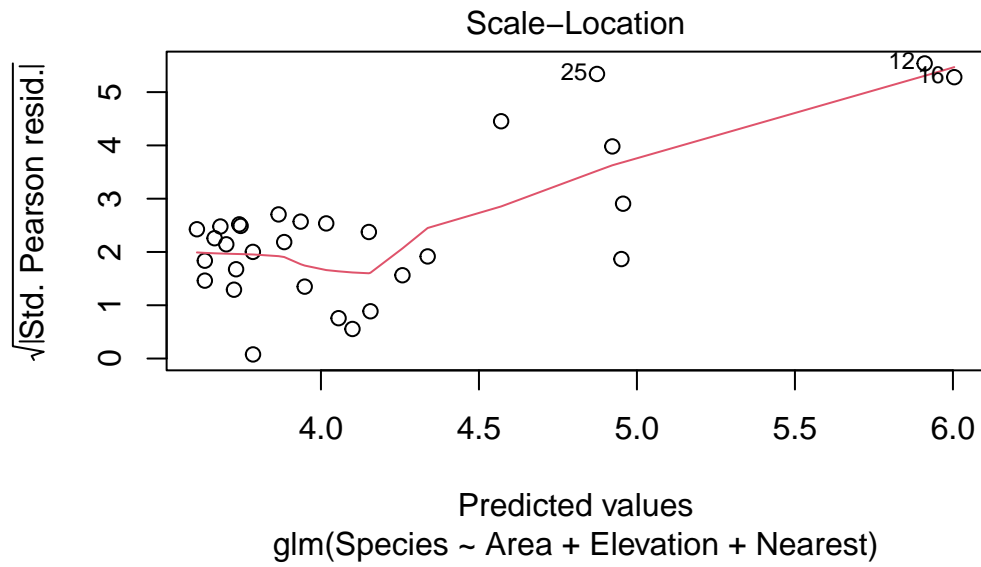
```
hist(rpois(nrow(gala),mean(gala$Species)))
```

Histogram of `rpois(nrow(gala), mean(gala$Species))`



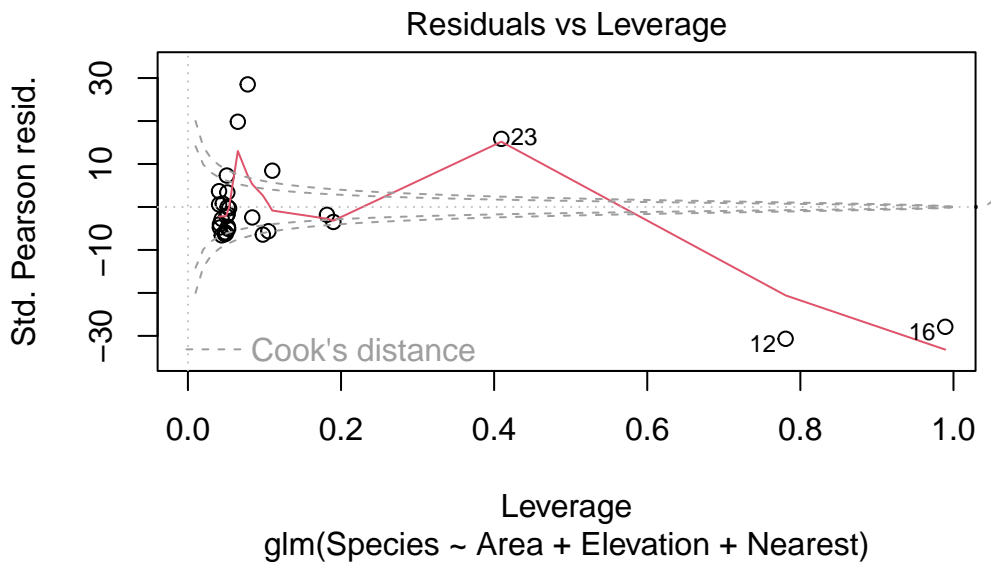
```
plot(modpl)
```





Warning in sqrt(crit * p * (1 - hh)/hh): NaNs produced

Warning in sqrt(crit * p * (1 - hh)/hh): NaNs produced



5. Introduction to linear mixed models

5.1. Lecture

5.1.1. Testing fixed effects

making a note that LRT on fixed effects should not be the preferred method and more importantly should be done using ML and not REML Fitsee pinheiro & Bates 2000 p76

5.1.2. Shrinkage

The following is an example of **shrinkage**, sometimes called **partial-pooling**, as it occurs in **mixed effects models**.

It is often the case that we have data such that observations are clustered in some way (e.g. repeated observations for units over time, students within schools, etc.). In mixed models, we obtain cluster-specific effects in addition to those for standard coefficients of our regression model. The former are called **random effects**, while the latter are typically referred to as **fixed effects** or **population-average** effects.

In other circumstances, we could ignore the clustering, and run a basic regression model. Unfortunately this assumes that all observations behave in the same way, i.e. that there are no cluster-specific effects, which would often be an untenable assumption. Another approach would be to run separate models for each cluster. However, aside from being problematic due to potentially small cluster sizes in common data settings, this ignores the fact that clusters are not isolated and potentially have some commonality.

Mixed models provide an alternative where we have cluster specific effects, but ‘borrow strength’ from the population-average effects. In general, this borrowing is more apparent for what would otherwise be more extreme clusters, and those that have less data. The following will demonstrate how shrinkage arises in different data situations.

5.1.2.1. Analysis

For the following we run a basic mixed model with a random intercept and random slopes for a single predictor variable. There are a number of ways to write such models, and the following does so for a single cluster c and observation i . y is a function of the covariate x , and otherwise we have a basic linear regression model. In this formulation, the random effects for a given cluster (u_{*c}) are added to each fixed effect (intercept b_0 and the effect of x , b_1). The random effects are multivariate normally distributed with some covariance. The per observation noise σ is assumed constant across observations.

$$\begin{aligned}\mu_{ic} &= (b_0 + u_{0c}) + (b_1 + u_{1c}) * x_{ic} \\ \mathbf{u}_0, \mathbf{u}_1 &\sim \mathcal{N}(0, \Sigma) \\ y &\sim \mathcal{N}(\mu, \sigma^2)\end{aligned}$$

Such models are highly flexible and have many extensions, but this simple model is enough for our purposes.

5.1.2.2. Data

Default settings for data creation are as follows:

- `obs_per_cluster` (observations per cluster) = 10
- `n_cluster` (number of clusters) = 100
- `intercept` (intercept) = 1
- `beta` (coefficient for x) = .5
- `sigma` (observation level standard deviation) = 1
- `sd_int` (standard deviation for intercept random effect)= .5
- `sd_slope` (standard deviation for x random effect)= .25
- `cor` (correlation of random effect) = 0
- `balanced` (fraction of overall sample size) = 1
- `seed` (for reproducibility) = 1024

In this setting, x is a standardized variable with mean zero and standard deviation of 1. Unless a fraction is provided for `balanced`, the N , i.e. the total sample size, is equal to `n_cluster * obs_per_cluster`. The following is the function that will be used to create the data, which tries to follow the model depiction above. It requires the `tidyverse` package to work.

5.1.2.3. Run the baseline model

We will use **lme4** to run the analysis. We can see that the model recovers the parameters fairly well, even with the default of only 1000 observations.

```
df <- create_data()

library(lme4)
mod <- lmer(y ~ x + (x | cluster), df)
summary(mod, cor = F)
```

```
Linear mixed model fit by REML. t-tests use Satterthwaite's method [
lmerModLmerTest]
```

```
Formula: y ~ x + (x | cluster)
Data: df
```

```
REML criterion at convergence: 3012.2
```

```
Scaled residuals:
```

```
      Min       1Q   Median       3Q      Max
-2.9392 -0.6352 -0.0061  0.6156  2.8721
```

```
Random effects:
```

| Groups | Name | Variance | Std.Dev. | Corr |
|----------|-------------|----------|----------|------|
| cluster | (Intercept) | 0.29138 | 0.5398 | |
| | x | 0.05986 | 0.2447 | 0.30 |
| Residual | | 0.99244 | 0.9962 | |

```
Number of obs: 1000, groups: cluster, 100
```

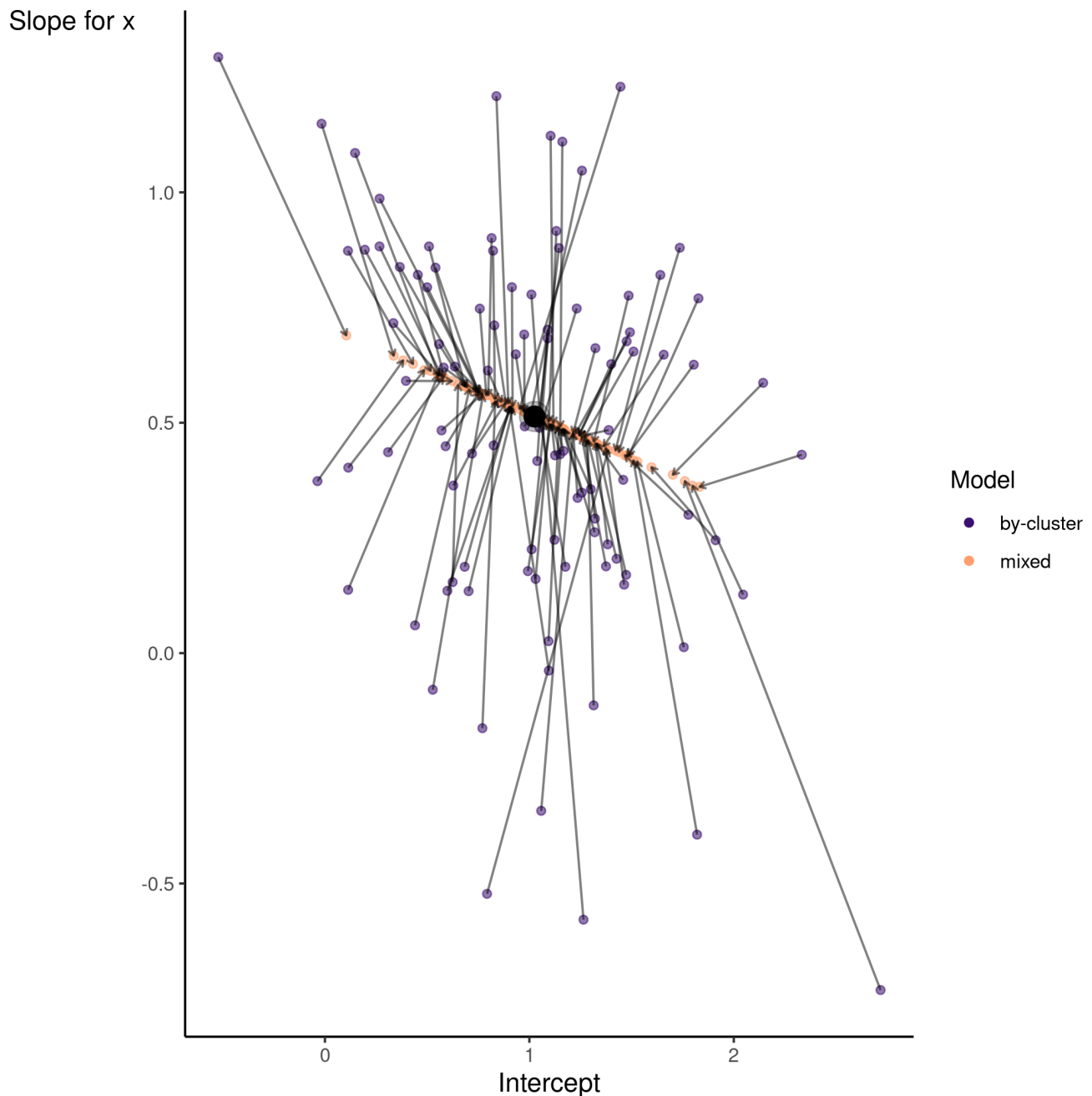
Fixed effects:

| | Estimate | Std. Error | df | t value | Pr(> t) | |
|-------------|----------|------------|----------|---------|----------|-----|
| (Intercept) | 0.93647 | 0.06282 | 98.38512 | 14.91 | <2e-16 | *** |
| x | 0.54405 | 0.04270 | 91.69469 | 12.74 | <2e-16 | *** |

 Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

5.1.2.4. Visualize the baseline model

Now it is time to visualize the results. We will use **gganimate** to bring the shrinkage into focus. We start with the estimates that would be obtained by a ‘regression-by-cluster’ approach or a linear regression for each cluster. The movement shown will be of those cluster-specific estimates toward the mixed model estimates. On the x axis is the estimate for the intercepts, on the y axis are the estimated slopes of the x covariate.



We see more clearly what the mixed model does. The general result is that cluster-specific effects (lighter color) are shrunk back toward the population-average effects (the ‘black hole’), as the imposed normal distribution for the random effects makes the extreme values less probable. Likewise, those more extreme cluster-specific effects, some of which are not displayed as they are so far from the population average, will generally have the most shrinkage imposed. In terms of prediction, it is akin to introducing bias for the cluster specific effects while lowering variance for prediction of new data, and allows us to make predictions on new categories we have not previously seen - we just assume an ‘average’ cluster effect, i.e. a random effect of 0.

5.1.2.5. Summary

Mixed models incorporate some amount of shrinkage for cluster-specific effects. Data nuances will determine the relative amount of ‘strength borrowed’, but in general, such models provide a good way for the data to speak for itself

when it should, and reflect an ‘average’ when there is little information. An additional benefit is that thinking about models in this way can be seen as a precursor to Bayesian approaches, which can allow for even more flexibility via priors, and more control over how shrinkage is added to the model.

5.2. Practical

5.2.1. Overview

This practical is intended to get you started fitting some simple mixed models with so called *random intercepts*. The tutorial is derived from one that accompanied the paper (Houslay and Wilson 2017), “Avoiding the misuse of BLUP in behavioral ecology”. Here, you will be working through a simplified version in which I have taken more time to cover the basic mixed models and don’t cover multivariate models which were really the main point of that paper. So if you find this material interesting don’t worry we will go through a more advanced version of the original paper on multivariate models in chapter XX. The original version will be worth a work through to help you break into multivariate mixed models anyway! Here we will:

- Learn how to fit - and interpret the results of - a simple univariate mixed effect model
- See how to add fixed and random effects to your model, and to test their significance in the normal frequentists sense

We are going to use the 📦 `lme4` (Bates et al. 2015) which is widely used and great for simple mixed models. However, since, for philosophical reasons, `lme4` does not provide any p-values for either fixed or random effects, we are going to use the 📦 `lmerTest` (Kuznetsova et al. 2017), which add a bunch a nice goodies to `lme4` For slightly more complex models, including multivariate ones, generalised models, and random effects of things like shared space, pedigree, phylogeny I tend to use different 📦 like `MCMCglmm` (Hadfield 2010) (which is Bayesian, look at Jarrod Hadfield’s excellent course notes (Hadfield 2010)) or `ASReml-R` (The VSNi Team 2023) (which is likelihood based/frequentist but sadly is not free).

5.2.2. R packages needed

First we load required libraries

```
library(lmerTest)
library(performance)
library(tidyverse)
library(rptR)
```

5.2.3. The superb wild unicorns of the Scottish Highlands

Unicorns, a legendary animal and also symbol of Scotland, are frequently described as extremely wild woodland creature but also a symbol of purity and grace. Here is one of most accurate representation of the legendary animal.



Figure 5.1.: The superb unicorn of the Scottish Highlands

Despite their image of purity and grace, unicorns (*Unicornus legendaricus*) are raging fighter when it comes to compete for the best sweets you can find at the bottom of rainbows (unicorn favourite source of food).

We want to know:

- If aggressiveness differs among individuals
- If aggressive behaviour is plastic (change with the environment)
- If aggressive behaviour depends on body condition of focal animal

With respect to plasticity, we will focus on rival size as an ‘environment’. Common sense, and animal-contest theory, suggest a small animal would be wise not to escalate an aggressive contest against a larger, stronger rival. However, there are reports in the legendary beastly literature that they get more aggressive as rival size increases. Those reports are based on small sample sizes and uncontrolled field observations by foreigners Munro baggers enjoying their whisky after a long day in the hills.

5.2.3.1. Experimental design

Here, we have measured aggression in a population of wild unicorns. We brought some ($n=80$) individual into the lab, tagged them so they were individually identifiable, then repeatedly observed their aggression when presented with model ‘intruders’ (animal care committee approved). There were three models; one of average unicorn (calculated as the population mean body length), one that was build to be 1 standard deviation below the population mean, and one that was 1 standard deviation above.

Data were collected on all individuals in two block of lab work. Within each block, each animal was tested 3 times, once against an ‘intruder’ of each size. The test order in which each animal experienced the three intruder sizes was randomised in each block. The body size of all focal individuals was measured at the beginning of each block so we know that too (and have two separate measures per individual).

5.2.3.2. looking at the data

Let's load the data file `unicorns_aggression.csv` in a R object named `unicorns` and make sure we understand what it contains

Solution

```
unicorns <- read.csv("data/unicorns_aggression.csv")
```

You can use `summary(unicorns)` to get an overview of the data and/or `str(unicorns)` to see the structure in the first few lines. This data frame has 6 variables:

```
str(unicorns)
```

```
'data.frame':  480 obs. of  6 variables:
 $ ID      : chr  "ID_1" "ID_1" "ID_1" "ID_1" ...
 $ block   : num  -0.5 -0.5 -0.5 0.5 0.5 0.5 -0.5 -0.5 -0.5 0.5 ...
 $ assay_rep : int  1 2 3 1 2 3 1 2 3 1 ...
 $ opp_size : int  -1 1 0 0 1 -1 1 -1 0 1 ...
 $ aggression: num  7.02 10.67 10.22 8.95 10.51 ...
 $ body_size : num  206 206 206 207 207 ...
```

```
summary(unicorns)
```

| ID | block | assay_rep | opp_size | aggression |
|------------------|---------------|------------|-------------|----------------|
| Length:480 | Min. :-0.5 | Min. :1 | Min. :-1 | Min. : 5.900 |
| Class :character | 1st Qu.: -0.5 | 1st Qu.: 1 | 1st Qu.: -1 | 1st Qu.: 8.158 |
| Mode :character | Median : 0.0 | Median : 2 | Median : 0 | Median : 8.950 |
| | Mean : 0.0 | Mean : 2 | Mean : 0 | Mean : 9.002 |
| | 3rd Qu.: 0.5 | 3rd Qu.: 3 | 3rd Qu.: 1 | 3rd Qu.: 9.822 |
| | Max. : 0.5 | Max. : 3 | Max. : 1 | Max. :12.170 |
| body_size | | | | |
| Min. :192.0 | | | | |
| 1st Qu.:229.7 | | | | |
| Median :250.0 | | | | |
| Mean :252.5 | | | | |
| 3rd Qu.:272.0 | | | | |
| Max. :345.2 | | | | |

So the different columns in the data set are:

- Individual **ID**
- Experimental **Block**, denoted for now as a continuous variable with possible values of -0.5 (first block) or +0.5 (second block)
- Individual **body_size**, as measured at the start of each block in kg

- The repeat number for each behavioural test, **assay_rep**
- Opponent size (**opp_size**), in standard deviations from the mean (i.e., -1,0,1)
- **aggression**, our behavioural trait, measured 6 times in total per individual (2 blocks of 3 tests)


maybe add something on how to look at data structure closely using tables

5.2.4. Do unicorns differ in aggressiveness? Your first mixed model

Fit a first mixed model with `lmer` that have only individual identity as a random effect and only a population mean.

Why, so simple? Because we simply want to partition variance around the mean into a component that among-individual variance and one that is within-individual variance.

! Important

We are going to use the function `lmer()` from the  `lme4` package. The notation of the model formula is similar as the notation for a linear model but now we also add random effects using the notation `(1 | r_effect)` which indicates that we want to fit the variable `r_effect` as a random effect for the intercept.

Thus, in `lmer` notation a simple model would be :

```
lmer(Y ~ x1 + x2 + (1 | r_effect), data = data)
```

💡 Solution

A sensible researcher would probably take the time to do some exploratory data plots here. So let's write a mixed model. This one is going to have no fixed effects except the mean, and just one random effect - individual identity.

```
m_1 <- lmer(aggression ~ 1 + (1 | ID), data = unicorns)
```

```
boundary (singular) fit: see help('isSingular')
```

There is a warning... something about "singularities". Ignore that for a moment.

Now you need to get the model output. By that I just mean use `summary(model_name)`.

💡 Solution

```
summary(m_1)
```

```
Linear mixed model fit by REML. t-tests use Satterthwaite's method [
lmerModLmerTest]
```

```
Formula: aggression ~ 1 + (1 | ID)
```

```
Data: unicorns
```

```
REML criterion at convergence: 1503.7
```

```
Scaled residuals:
```

```

      Min       1Q   Median       3Q      Max
-2.68530 -0.73094 -0.04486  0.71048  2.74276

Random effects:
 Groups   Name                Variance Std.Dev.
 ID       (Intercept)  0.000    0.000
 Residual                    1.334    1.155
Number of obs: 480, groups:  ID, 80

Fixed effects:
              Estimate Std. Error      df t value Pr(>|t|)
(Intercept)   9.00181    0.05272 479.00000   170.7  <2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
optimizer (nloptwrap) convergence code: 0 (OK)
boundary (singular) fit: see help('isSingular')

```

In the summary you will find a table of fixed effects.

```

Fixed effects:
              Estimate Std. Error      df t value Pr(>|t|)
(Intercept)   9.00181    0.05272 479.00000   170.7  <2e-16 ***

```

The intercept (here the mean) is about 9 and is significantly >0 - fine, but not very interesting to us.

You will also find a random effect table that contains estimates of the among individual (ID) and residual variances.

```


Random effects:
 Groups   Name                Variance Std.Dev.
 ID       (Intercept)  0.000    0.000
 Residual                    1.334    1.155
Number of obs: 480, groups:  ID, 80

```

The among individual (ID) is estimated as zero. In fact this is what the cryptic warning was about: in most situations the idea of a random effect explaining less than zero variance is not sensible (strangely there are exception!). So by default the variance estimates are constrained to lie in positive parameter space. Here in trying to find the maximum likelihood solution for among-individual variance, our model has run up against this constraint.

5.2.4.1. Testing for random effects

We can test the statistical significance of the random effect using the `ranova()` command in `lmerTest`. This function is actually doing a *likelihood ratio test* (LRT) of the random effect. The premise of which is that twice the difference in log-likelihood of the full and reduced (i.e. with the random effect dropped) is itself distributed as χ^2 with DF equal to the number of parameters dropped (here 1). Actually, there is a good argument that this is too conservative, but we can discuss that later. So let's do the LRT for the random effect using `ranova()`

 Solution

```
ranova(m_1)
```

ANOVA-like table for random-effects: Single term deletions

Model:

```
aggression ~ (1 | ID)
      npar  logLik    AIC LRT Df Pr(>Chisq)
<none>    3 -751.83 1509.7
(1 | ID)  2 -751.83 1507.7  0  1          1
```

There is apparently no among-individual variance in aggressiveness.

So this is a fairly rubbish and underwhelming model. Let's improve it.

5.2.5. Do unicorns differ in aggressiveness? A better mixed model

The answer we got from our first model is **not** wrong, it estimated the parameters we asked for and that might be informative or not and that might be representative or not of the true biology. Anyway all models are **wrong** but as models go this one is fairly rubbish. In fact we have explained no variation at all as we have no fixed effects (except the mean) and our random effect variance is zero. We would have seen just how pointless this model was if we'd plotted it

```
plot(m_1)
```

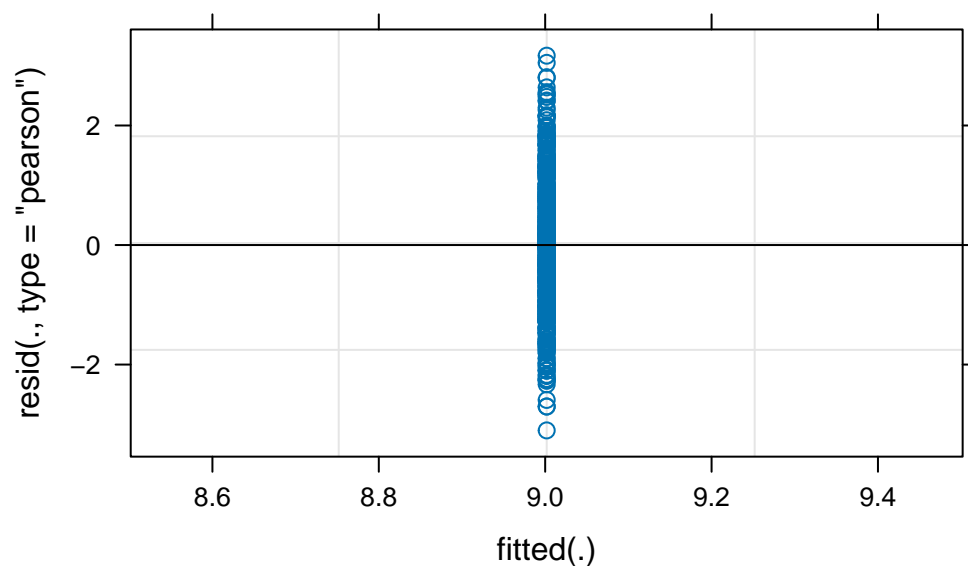



Figure 5.2.: Fitted values vs residuals for a simple mixed model of unicorn aggression

So we can probably do better at modelling the data, which may or may not change our view on whether there is any real variation among unicorns in aggressiveness.

For instance, we can (and should have started with) an initial plot of the phenotypic data against opponent size indicates to have a look at our prediction.

💡 Solution

The code below uses the excellent  `ggplot2` but the same figure can be done using base R code.

```
ggplot(unicorns, aes(x = opp_size, y = aggression)) +
  geom_jitter(
    alpha = 0.5,
    width = 0.05
  ) +
  scale_x_continuous(breaks = c(-1, 0, 1)) +
  labs(
    x = "Opponent size (SD)",
    y = "Aggression"
  ) +
  theme_classic()
```

```
ggplot(unicorns, aes(x = opp_size, y = aggression)) +
  geom_jitter(
    alpha = 0.5,
    width = 0.05
  ) +
  scale_x_continuous(breaks = c(-1, 0, 1)) +
  labs(
    x = "Opponent size (SD)",
    y = "Aggression"
  ) +
  theme_classic()
```

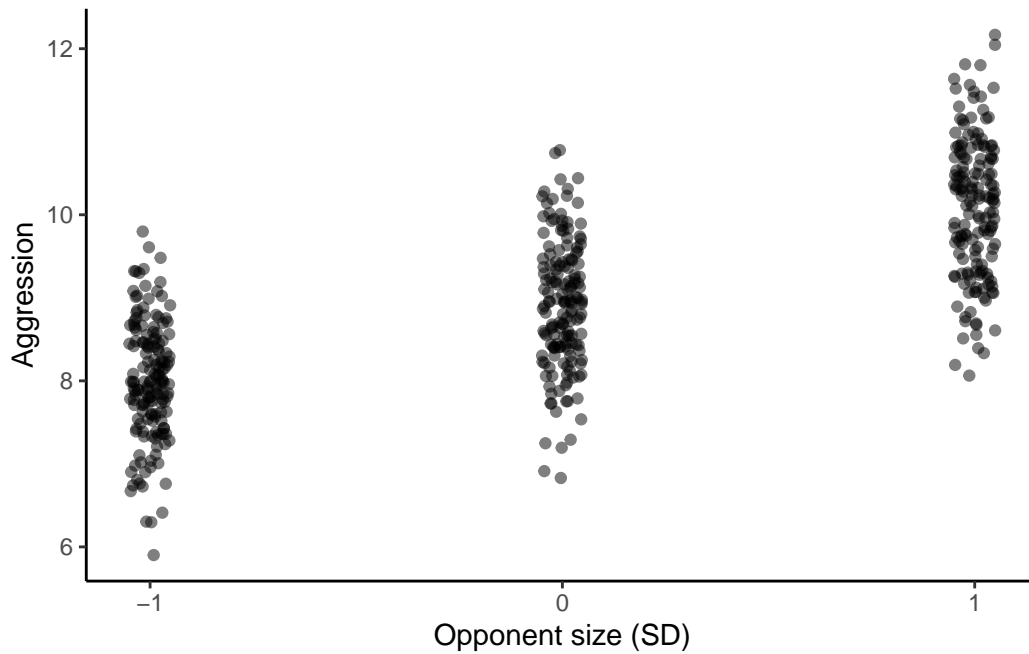


Figure 5.3.: Unicorn aggressivity as a function of opponent size when fighting for sweets

As predicted, there is a general increase in aggression with opponent size (points are lightly jittered on the x-axis to show the spread of data a little better)

You can see the same thing from a quick look at the population means for aggression at opponent size. Here we do it with the `kable` function that makes nice tables in `rmarkdown` documents.


```
unicorns %>%
  group_by(opp_size) %>%
  summarise(mean_aggr = mean(aggression)) %>%
  knitr::kable(digits = 2)
```

| opp_size | mean_aggr |
|----------|-----------|
| -1 | 8.00 |
| 0 | 8.91 |
| 1 | 10.09 |

So, there does appear to be plasticity of aggression with changing size of the model opponent. But other things may explain variation in aggressiveness too - what about block for instance? Block effects may not be the subject of any biologically interesting hypotheses, but accounting for any differences between blocks could remove noise.

There may also be systematic change in behaviour as an individual experiences more repeat observations (i.e. exposure to the model). Do they get sensitised or habituated to the model intruder for example?

So let's run a mixed model with the same random effect of individual, but with a fixed effects of opponent size (our predictor of interest) and experimental block.


 Solution

```
m_2 <- lmer(aggression ~ opp_size + block + (1 | ID), data = unicorns)
```

5.2.5.1. Diagnostic plots

Run a few diagnostic plots before we look at the answers. In diagnostic plots, we want to check the condition of applications of the linear mixed model which are the same 4 as the linear model plus 2 extra:

1. Linearity of the relation between covariates and the response

 Solution

Done with data exploration graph (i.e. just plot the data see if it is linear) - see previous graph @ref(fig:rplotaggr).

2. No error on measurement of covariates

 Solution

assumed to be correct if measurement error is lower than 10% of variance in the variable - I know this sounds pretty bad

3. Residual have a Gaussian distribution

 Solution

using quantile-quantile plot or histogram of residuals

```
par(mfrow = c(1, 2)) # multiple graphs in a window
qqnorm(residuals(m_2)) # a q-q plot
qqline(residuals(m_2))
hist(resid(m_2)) # are the residuals roughly Gaussian?
```

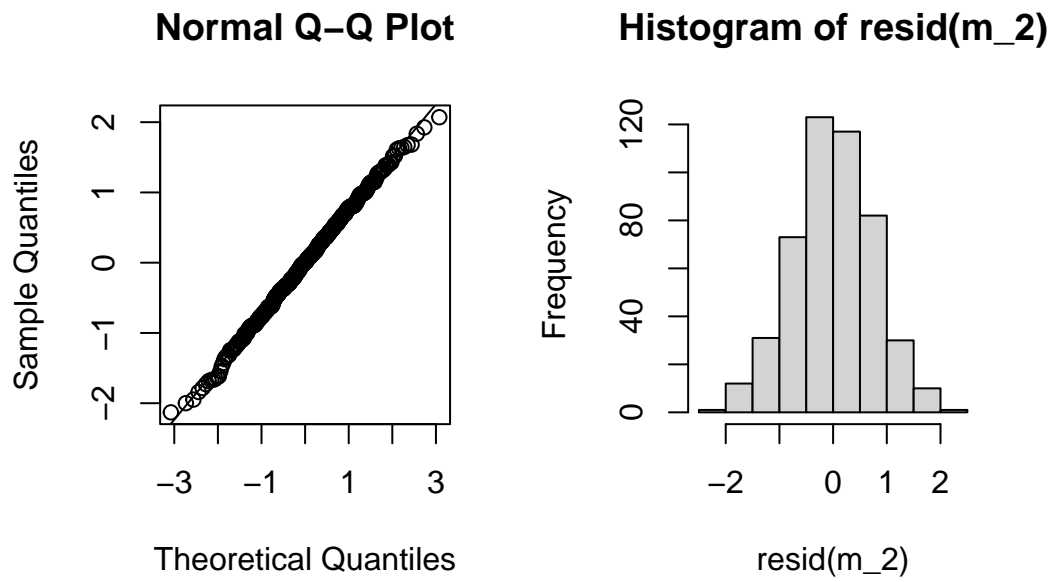


Figure 5.4.: Checking residuals have Gaussian distribution

4. Homoscedasticity (variance of residuals is constant across covariates)

💡 Solution

Using plot of residuals by fitted values

```
plot(m_2)
```

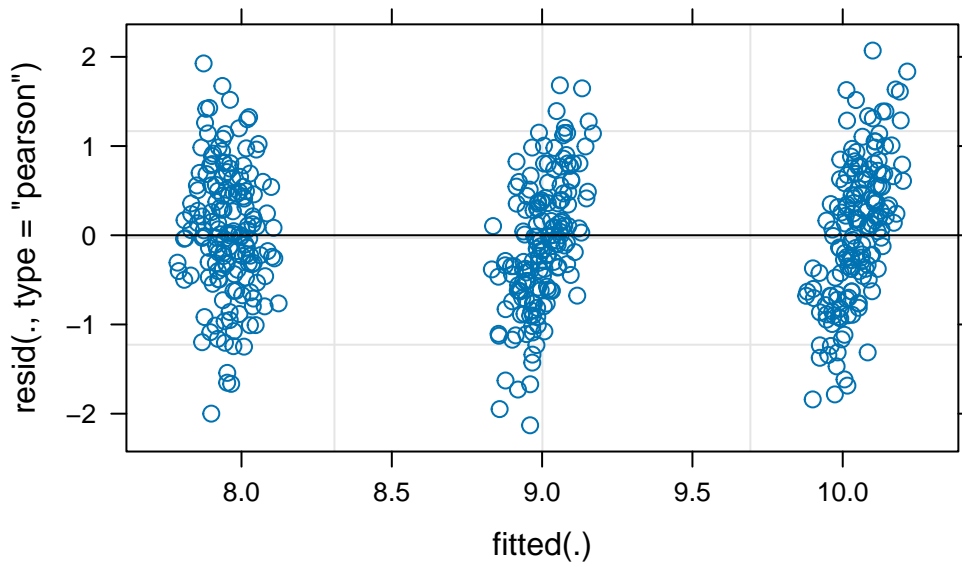


Figure 5.5.: Residuals by fitted values for model m_2 to check homoscedasticity

5. Random effects have a Gaussian distribution

💡 Solution

histogram of the predictions for the random effects (BLUPs)

```
# extracting blups
r1 <- as.data.frame(ranef(m_2, condVar = TRUE))
par(mfrow = c(1, 2))
hist(r1$condval)
qqnorm(r1$condval)
qqline(r1$condval)
```

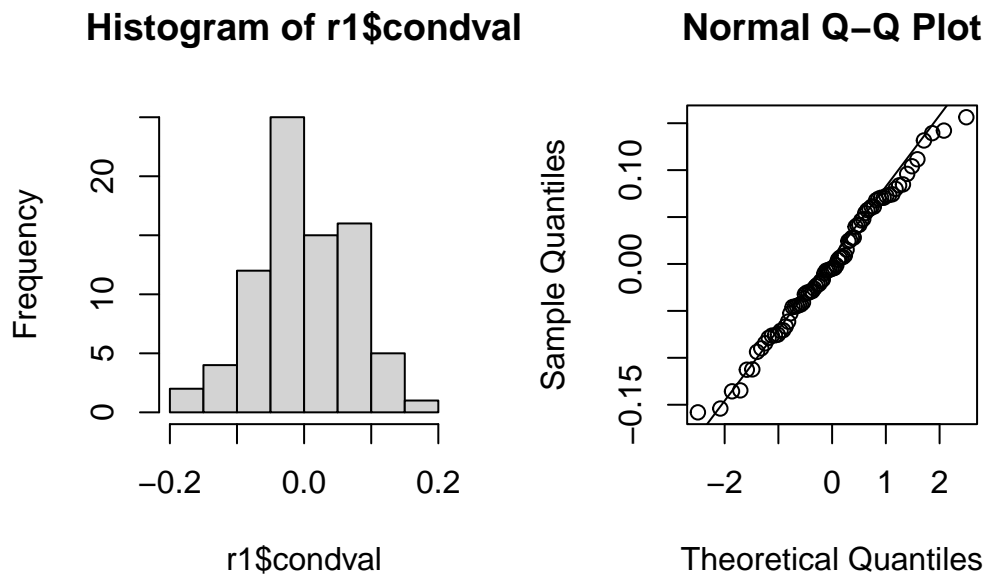


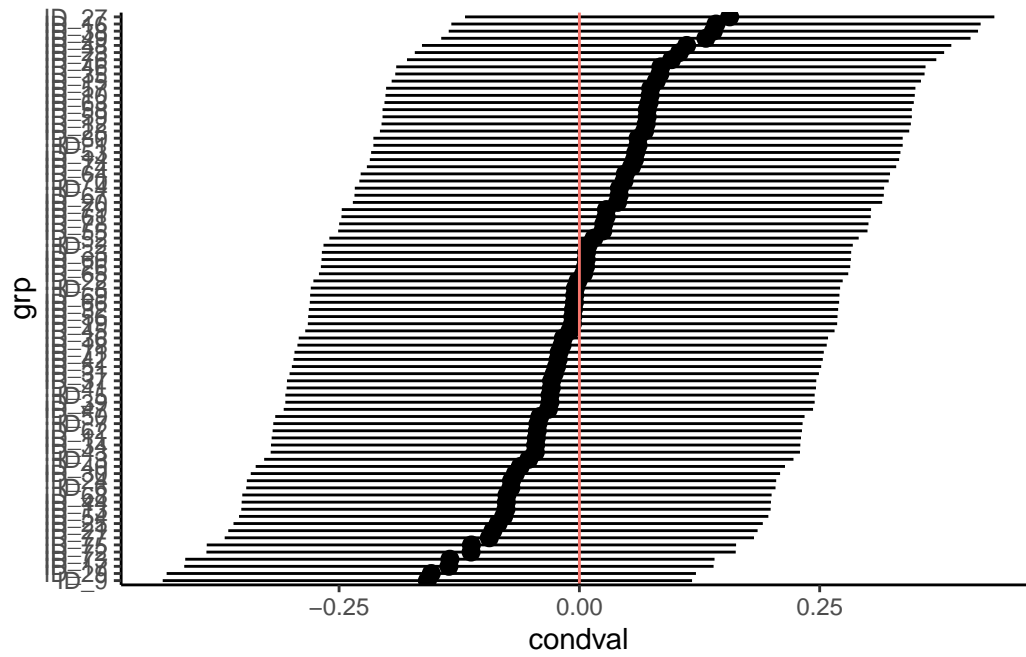
Figure 5.6.: Checking random effects are gaussian

6. Residual variance is constant across all levels of a random effect

💡 Solution

No straightforward solution to deal with that. We can just do a plot is absolutely not-informative for that problem but I always like to look at. It is the plot of the sorted BLUPs with their associated errors.

```
r1 <- r1[order(r1$condval), ] # sorting the BLUPs
ggplot(r1, aes(y = grp, x = condval)) +
  geom_point() +
  geom_pointrange(
    aes(xmin = condval - condsd * 1.96, xmax = condval + condsd * 1.96)
  ) +
  geom_vline(aes(xintercept = 0, color = "red")) +
  theme_classic() +
  theme(legend.position = "none")
```



Here is a great magic trick 🌟 because 3-5 and more can be done in one step

💡 Solution

You need to use the function `check_model()` from the 📦 `performance` package.

```
check_model(m_2)
```

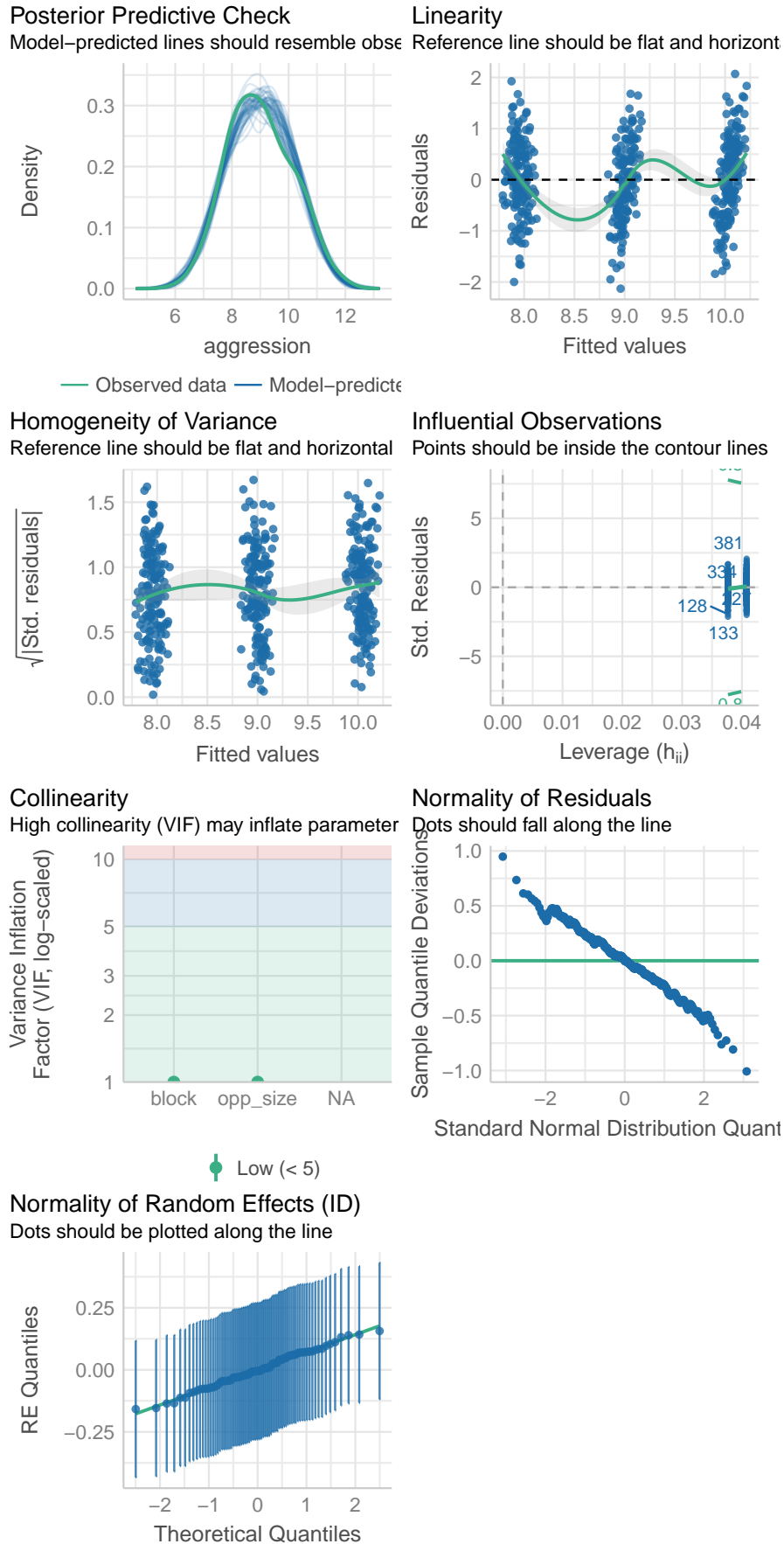



Figure 5.7.: Graphical check of model assumptions

5.2.5.2. Inferences

Now summarise this model. We will pause here for you to think about and discuss a few things: * What can you take from the fixed effect table? * How do you interpret the intercept now that there are other effects in the model? * What would happen if we scaled our fixed covariates differently? Why?

 Solution

```
summary(m_2)
```

```
Linear mixed model fit by REML. t-tests use Satterthwaite's method [
lmerModLmerTest]
```

```
Formula: aggression ~ opp_size + block + (1 | ID)
Data: unicorns
```

```
REML criterion at convergence: 1129.9
```

```
Scaled residuals:
```

```
      Min       1Q   Median       3Q      Max
-2.79296 -0.64761  0.00155  0.67586  2.71456
```

```
Random effects:
```

```
Groups   Name             Variance Std.Dev.
ID       (Intercept)  0.02478  0.1574
Residual                    0.58166  0.7627
```

```
Number of obs: 480, groups: ID, 80
```

```
Fixed effects:
```

```
              Estimate Std. Error    df t value Pr(>|t|)
(Intercept)   9.00181    0.03901  79.00000  230.778 <2e-16 ***
opp_size      1.04562    0.04263  398.00000   24.525 <2e-16 ***
block        -0.02179    0.06962  398.00000   -0.313  0.754
```

```
---
```

```
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
Correlation of Fixed Effects:
```

```
      (Intr) opp_sz
opp_size 0.000
block    0.000 0.000
```

 Exercise

Try tweaking the fixed part of your model:

- What happens if you add more fixed effects? Try it!
- Could focal body size also matter? If so, should you rescale before adding it to the model?
- Should you add interactions (e.g. block:opp_size)?
- Should you drop non-significant fixed effects?

 Exercise**Having changed the fixed part of your model, do the variance estimates change at all?**

- Is among-individual variance always estimated as zero regardless of fixed effects?
- Is among-individual variance significant with some fixed effects structures but not others?

5.2.6. What is the repeatability?

As a reminder, repeatability is the proportion of variance explained by a random effect and it is estimate as the ratio of the variance associated to a random effect by the total variance, or the sum of the residual variance and the different variance component associated with the random effects. In our first model among-individual variance was zero, so R was zero. If we have a different model of aggression and get a non-zero value of the random effect variance, we can obviously calculate a repeatability estimate (R). So we are all working from the same starting point, let's all stick with a common set of fixed effects from here on:

```
m_3 <- lmer(
  aggression ~ opp_size + scale(body_size, center = TRUE, scale = TRUE)
  + scale(assay_rep, scale = FALSE) + block
  + (1 | ID),
  data = unicorns
)
summary(m_3)
```

```
Linear mixed model fit by REML. t-tests use Satterthwaite's method [
lmerModLmerTest]
Formula:
aggression ~ opp_size + scale(body_size, center = TRUE, scale = TRUE) +
  scale(assay_rep, scale = FALSE) + block + (1 | ID)
Data: unicorns
```

```
REML criterion at convergence: 1136.5
```

```
Scaled residuals:
   Min       1Q   Median       3Q      Max
-2.85473 -0.62831  0.02545  0.68998  2.74064
```

```
Random effects:
 Groups   Name      Variance Std.Dev.
 ID       (Intercept) 0.02538  0.1593
 Residual                0.58048  0.7619
Number of obs: 480, groups: ID, 80
```

```
Fixed effects:
              Estimate Std. Error    df
(Intercept)    9.00181    0.03907  78.07315
opp_size        1.05141    0.04281 396.99857
```



```

scale(body_size, center = TRUE, scale = TRUE) 0.03310 0.03896 84.21144
scale(assay_rep, scale = FALSE) -0.05783 0.04281 396.99857
block -0.02166 0.06955 397.00209
t value Pr(>|t|)
(Intercept) 230.395 <2e-16 ***
opp_size 24.562 <2e-16 ***
scale(body_size, center = TRUE, scale = TRUE) 0.850 0.398
scale(assay_rep, scale = FALSE) -1.351 0.177
block -0.311 0.756
---
Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

```

Correlation of Fixed Effects:

```

(Intr) opp_sz sc=Ts=T s(_s=F
opp_size 0.000
s(_c=TRs=T 0.000 0.000
s(_s=FALSE 0.000 -0.100 0.000
block 0.000 0.000 0.002 0.000

```

So we'd probably calculate R using the individual and residual variance simply as:

```
0.02538 / (0.02538 + 0.58048)
```

```
[1] 0.04189087
```

Exercise

Do you see where I took the numbers ?

We can use some more fancy coding to extract the estimates and plugged them in a formula to estimate the repeatability

```

v_id <- VarCorr(m_3)$ID[1, 1]
v_r <- attr(VarCorr(m_3), "sc")^2
r_man <- v_id / (v_id + v_r)
r_man

```

```
[1] 0.04188879
```

Which yields an estimate of approximately $R=4\%$. Strictly speaking we should make clear this a **conditional repeatability** estimate.

Conditional on what you might ask... on the fixed effects in your model. So our best estimate of 4% refers to the proportion of variance in aggressiveness *not explained by fixed effects* that is explained by individual identity. This isn't much and still won't be significant, but illustrates the point that conditional repeatabilities often have a tendency to go up as people explain more of the residual variance by adding fixed effects. This is fine and proper, but can mislead the unwary reader. It also means that decisions about which fixed effects to include in your model need to be based on how you want to interpret R not just on, for instance, whether fixed effects are deemed significant.

5.2.7. A quick note on uncertainty

Using `lmer` in the `lme4` there isn't a really simple way to put some measure of uncertainty (SE or CI) on derived parameters like repeatabilities. This is a bit annoying. Such things are more easily done with other mixed model like `MCMCglmm` and `asreml` which are a bit more specialist. If you are using `lmer` for models you want to publish then you could look into the `rptR` (Stoffel et al. 2017). This acts as a 'wrapper' for `lmer` models and adds some nice functionality including options to bootstrap confidence intervals. Regardless, of how you do it, if you want to put a repeatability in one of your papers as a key result - it really should be accompanied by a measure of uncertainty just like any other effect size estimate.

Here I am estimating the repeatability and using bootstrap to estimate a confidence interval and a probability associated with the repeatability with the `rptR`. For more information about the use of the package and the theory behind it suggest the excellent paper associated with the package (Stoffel et al. 2017)

```
r_rpt <- rptGaussian(
  aggression ~ opp_size + block + (1 | ID),
  grname = "ID", data = unicorns
)
```

Bootstrap Progress:

```
r_rpt
```

Repeatability estimation using the lmm method

```
Repeatability for ID
R = 0.041
SE = 0.03
CI = [0, 0.103]
P = 0.0966 [LRT]
  NA [Permutation]
```

5.2.8. An easy way to mess up your mixed models

We will try some more advanced mixed models in a moment to explore plasticity in aggressiveness a bit more. First let's quickly look for among-individual variance in focal body size. Why not? We have the data handy, everyone says morphological traits are very repeatable and - lets be honest - who wouldn't like to see a small P value after striking out with aggressiveness.

Include a random effect of ID as before and maybe a fixed effect of block, just to see if the beasties were growing a bit between data collection periods.

```
lmer_size <- lmer(body_size ~ block + (1 | ID),
  data = unicorns
)
```

Summarise and test the random effect.

💡 Solution

```
summary(lmer_size)
```

```
Linear mixed model fit by REML. t-tests use Satterthwaite's method [
lmerModLmerTest]
Formula: body_size ~ block + (1 | ID)
Data: unicorns
```

```
REML criterion at convergence: 3460.7
```

```
Scaled residuals:
```

```
      Min       1Q   Median       3Q      Max
-1.80452 -0.71319  0.00718  0.70280  1.81747
```

```
Random effects:
```

```
Groups   Name          Variance Std.Dev.
ID       (Intercept) 936.01   30.594
Residual                34.32    5.858
```

```
Number of obs: 480, groups: ID, 80
```

```
Fixed effects:
```

```
              Estimate Std. Error      df t value Pr(>|t|)
(Intercept) 252.5031     3.4310  79.0000  73.595 <2e-16 ***
block        -0.1188     0.5348 399.0000  -0.222  0.824
```

```
---
```

```
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
Correlation of Fixed Effects:
```

```
  (Intr)
block 0.000
```

```
ranova(lmer_size)
```

```
ANOVA-like table for random-effects: Single term deletions
```

```
Model:
```

```
body_size ~ block + (1 | ID)
```

```

      npar  logLik    AIC    LRT Df Pr(>Chisq)
<none>    4 -1730.4 3468.7
(1 | ID)   3 -2325.6 4657.1 1190.4  1 < 2.2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

```

Exercise

What might you conclude, and why would this be foolish?

Solution

Hopefully you spotted the problem here. You have fed in a data set with 6 records per individual (with 2 sets of 3 identical values per unicorns), when you know size was only measured twice in reality. This means you'd expect to get a (potentially very) upwardly biased estimate of R and a (potentially very) downwardly biased P value when testing among-individual variance.

Exercise

How can we do it properly?

Solution

We can prune the data to the two actual observations per unicorns by just selecting the first assay in each block.

```

unicorns2 <- unicorns[unicorns$assay_rep == 1, ]

lmer_size2 <- lmer(body_size ~ block + (1 | ID),
  data = unicorns2
)
summary(lmer_size2)

```

```

Linear mixed model fit by REML. t-tests use Satterthwaite's method [
lmerModLmerTest]

```

```

Formula: body_size ~ block + (1 | ID)

```

```

Data: unicorns2

```

```

REML criterion at convergence: 1373.4

```

```

Scaled residuals:

```

```

      Min       1Q   Median       3Q      Max
-1.54633 -0.56198  0.01319  0.56094  1.42095

```

```

Random effects:

```

```

Groups   Name             Variance Std.Dev.
ID       (Intercept)  912.84  30.213
Residual                    57.78   7.601

```

```

Number of obs: 160, groups: ID, 80

Fixed effects:
              Estimate Std. Error      df t value Pr(>|t|)
(Intercept) 252.5031     3.4310   79.0000  73.595   <2e-16 ***
block       -0.1188     1.2019   79.0000  -0.099    0.922
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Correlation of Fixed Effects:
      (Intr)
block 0.000

```

```
ranova(lmer_size2)
```

ANOVA-like table for random-effects: Single term deletions

```

Model:
body_size ~ block + (1 | ID)
              npar  logLik   AIC   LRT Df Pr(>Chisq)
<none>         4 -686.68 1381.3
(1 | ID)       3 -771.93 1549.9 170.51  1 < 2.2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

```

Summarise and test your random effect and you'll see the qualitative conclusions will actually be very similar using the pruned data set. Of course this won't generally be true, so just be careful. Mixed models are intended to help you model repeated measures data with non-independence, but they won't get you out of trouble if you mis-represent the true structure of observations on your dependent variable.

5.2.9. Happy mixed-modelling



Figure 5.8.: The superb unicorn

6. Introduction to GLMM

6.1. Lecture

theoretical intro to glmm and introduce DHarma package to evaluate fit of glmm



Figure 6.1.: Dream pet dragon

6.2. Practical

This is an adapted version largely inspired by the tutorial in (Bolker et al. 2009). Spatial variation in nutrient availability and herbivory is likely to cause population differentiation and maintain genetic diversity in plant populations. Here we measure the extent to which mouse-ear cress (*Arabidopsis thaliana*) exhibits population and genotypic variation in their responses to these important environmental factors. We are particularly interested in whether these populations exhibit nutrient mediated compensation, where higher nutrient levels allow genotypes to better tolerate herbivory (Banta et al. 2010). We use GLMMs to estimate the effect of nutrient levels, simulated herbivory, and their interaction on fruit production in *Arabidopsis thaliana* (fixed effects), and the extent to which populations vary in their responses (random effects, or variance components)

6.2.1. Packages and functions

You need to download the “extra_funs.R” script for some functions used in the Practical

```
library(lme4)
library(tidyverse)
library(patchwork)
library(lattice)
library(DHARMa)
source("data/extra_funs.R")
```

6.2.2. The data set

In this data set, the response variable is the number of fruits (i.e. seed capsules) per plant. The number of fruits produced by an individual plant (the experimental unit) was hypothesized to be a function of fixed effects, including nutrient levels (low vs. high), simulated herbivory (none vs. apical meristem damage), region (Sweden, Netherlands, Spain), and interactions among these. Fruit number was also a function of random effects including both the population and individual genotype. Because *Arabidopsis* is highly selfing, seeds of a single individual served as replicates of that individual. There were also nuisance variables, including the placement of the plant in the greenhouse, and the method used to germinate seeds. These were estimated as fixed effects but interactions were excluded.

- `X` observation number (we will use this observation number later, when we are accounting for overdispersion)
- `reg` a factor for region (Netherlands, Spain, Sweden).
- `popu` a factor with a level for each population.
- `gen` a factor with a level for each genotype.
- `rack` a nuisance factor for one of two greenhouse racks.
- `nutrient` a factor with levels for minimal or additional nutrients.
- `amd` a factor with levels for no damage or simulated herbivory (apical meristem damage; we will sometimes refer to this as “clipping”)
- `status` a nuisance factor for germination method.
- `total.fruits` the response; an integer count of the number of fruits per plant.

6.2.3. Specifying fixed and random Effects


Here we need to select a realistic full model, based on the scientific questions and the data actually at hand. We first load the data set and make sure that each variable is appropriately designated as numeric or factor (i.e. categorical variable).

```
dat_tf <- read.csv("data/Banta_TotalFruits.csv")
str(dat_tf)
```

```
'data.frame':  625 obs. of  9 variables:
 $ X           : int  1 2 3 4 5 6 7 8 9 10 ...
 $ reg         : chr  "NL" "NL" "NL" "NL" ...
 $ popu        : chr  "3.NL" "3.NL" "3.NL" "3.NL" ...
 $ gen         : int  4 4 4 4 4 4 4 4 4 5 ...
 $ rack        : int  2 1 1 2 2 2 2 1 2 1 ...
 $ nutrient    : int  1 1 1 1 8 1 1 1 8 1 ...
```



```
$ amd          : chr  "clipped" "clipped" "clipped" "clipped" ...
$ status       : chr  "Transplant" "Petri.Plate" "Normal" "Normal" ...
$ total.fruits: int  0 0 0 0 0 0 0 3 2 0 ...
```

The X, gen, rack and nutrient variables are coded as integers, but we want them to be factors. We use `mutate()` `dplyr` , which operates within the data set, to avoid typing lots of commands like `dat_tf$rack <- factor(dat_tf$rack)`. At the same time, we reorder the clipping variable so that "unclipped" is the reference level (we could also have used `relevel(amd, "unclipped")`).

```
dat_tf <- mutate(
  dat_tf,
  X = factor(X),
  gen = factor(gen),
  rack = factor(rack),
  amd = factor(amd, levels = c("unclipped", "clipped")),
  nutrient = factor(nutrient, label = c("Low", "High"))
)
```

Now we check replication for each genotype (columns) within each population (rows).

```
(reptab <- with(dat_tf, table(popu, gen)))
```

| | gen | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|-----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|---|---|---|---|---|---|---|---|---|---|---|----|----|----|---|---|---|
| popu | 4 | 5 | 6 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 27 | 28 | 30 | 34 | 35 | 36 | | | | | | | | | | | | | | | | | |
| 1.SP | 0 | 0 | 0 | 0 | 0 | 39 | 26 | 35 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | | |
| 1.SW | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 28 | 20 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |
| 2.SW | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 18 | 14 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |
| 3.NL | 31 | 11 | 13 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |
| 5.NL | 0 | 0 | 0 | 35 | 26 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |
| 5.SP | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 43 | 22 | 12 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 6.SP | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 13 | 24 | 14 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 7.SW | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 45 | 47 | 45 | 0 | 0 | |
| 8.SP | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 13 | 16 | 35 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Exercise

Exercise: this mode of inspection is OK for this data set but might fail for much larger data sets or for more levels of nesting. See if you can think of some other numerical or graphical methods for inspecting the structure of data sets.

1. `plot(reptab)` gives a mosaic plot of the two-way table; examine this, see if you can figure out how to interpret it, and decide whether you think it might be useful
2. try the commands `colSums(reptab>0)` (and the equivalent for `rowSums`) and figure out what they are telling you.
3. Using this recipe, how would you compute the range of number of genotypes per treatment combination?

💡 Solution

1. Do you find the mosaic plot you obtained ugly and super hard to read? Me too 😊

```
plot(reptab)
```

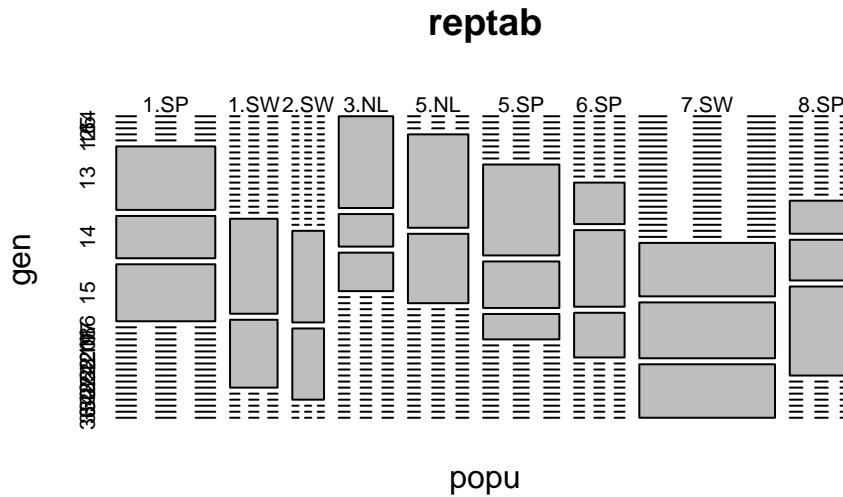


Figure 6.2.: A truly useless plot no one can understand

2. `colSums()` do the sum of all the rows for each columns of a table. So `colSums(reptab>0)` gives you for each genotype the number of populations (lines) where you have at least 1 observations.

```
colSums(reptab > 0)
```

```
4  5  6 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 27 28 30 34 35 36
1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1
```

```
rowSums(reptab > 0)
```

```
1.SP 1.SW 2.SW 3.NL 5.NL 5.SP 6.SP 7.SW 8.SP
  3    2    2    3    2    3    3    3    3
```

3. You firsts need to create a new table of number of observations per treatment and genotypes

```
reptab2 <- with(dat_tf, table(paste(amd, nutrient, sep = "_"), gen))
range(reptab2)
```

```
[1] 2 13
```

This reveals that we have only 2–4 populations per region and 2–3 genotypes per population. However, we also have 2–13 replicates per genotype for each treatment combination (four unique treatment combinations: 2 levels of nutrients by 2 levels of simulated herbivory). Thus, even though this was a reasonably large experiment (625 plants), there were a very small number of replicates with which to estimate variance components, and many more potential interactions than our data can support. Therefore, judicious selection of model terms, based on both biology and the data, is warranted. We note that we don't really have enough levels per random effect, nor enough replication per unique treatment combination. Therefore, we decide to omit the fixed effect of “region”, although we recognize that populations in different regions are widely geographically separated.

However, as in all GLMMs where the scale parameter is treated as fixed and deviations from the fixed scale parameter would be identifiable (i.e. Poisson and binomial ($N > 1$), but not binary, models) we may have to deal with overdispersion.

6.2.4. Look at overall patterns in data

I usually like to start with a relatively simple overall plot of the data, disregarding the random factors, just to see what's going on. For reasons to be discussed below, we choose to look at the data on the log (or $\log(1 + x)$ scale. Let's plot either box-and-whisker plots (useful summaries) or dot plots (more detailed, good for seeing if we missed anything).

Warning: `qplot()` was deprecated in ggplot2 3.4.0.

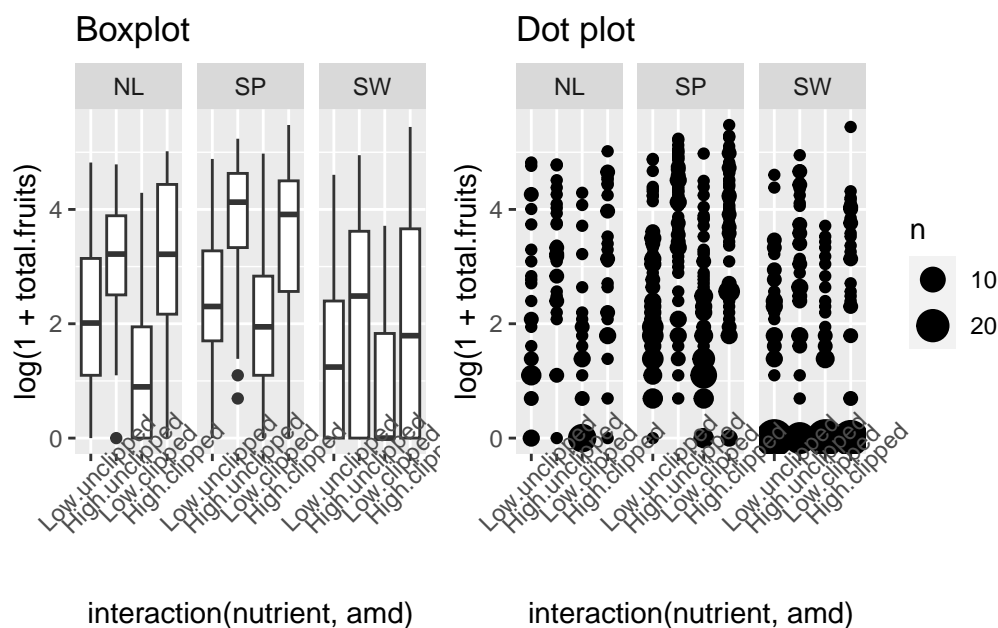




Figure 6.3.: Number of fruits ($\log + 1$) as a function of treatments

 Exercise

Exercise generate these plots and figure out how they work before continuing. Try conditioning/faceting on population rather than region: for `facet_wrap` you might want to take out the `nrow = 1` specification. If you want try reorder the subplots by overall mean fruit set and/or colour the points according to the region they come from.

 Solution

```
p1 <- qplot(
  interaction(nutrient, amd),
  log(1 + total.fruits),
  data = dat_tf, geom = "boxplot") +
  facet_wrap(~reg, nrow = 1) +
  theme(axis.text.x = element_text(angle = 45)) +
  ggtitle("Boxplot")
p2 <- qplot(
  interaction(nutrient, amd),
  log(1 + total.fruits),
  data = dat_tf) +
  facet_wrap(~reg, nrow = 1) +
  stat_sum() +
  theme(axis.text.x = element_text(angle = 45)) +
  ggtitle("Dot plot")
p1 + p2
```

6.2.5. Choose an error distribution

The data are non-normal in principle (i.e., count data, so our first guess would be a Poisson distribution). If we transform total fruits with the canonical link function (log), we hope to see relatively homogeneous variances across categories and groups.

First we define a new factor that represents every combination of genotype and treatment (nutrient \times clipping) treatment, and sort it in order of increasing mean fruit set.

```
dat_tf <- dat_tf %>%
  mutate(
    gna = reorder(interaction(gen, nutrient, amd), total.fruits, mean)
  )
```

Now time to plot it

```
ggplot(dat_tf, aes(x = gna, y = log(1 + total.fruits))) +
  geom_boxplot() +
```

```
theme_bw() +
theme(axis.text.x = element_text(angle = 90))
```

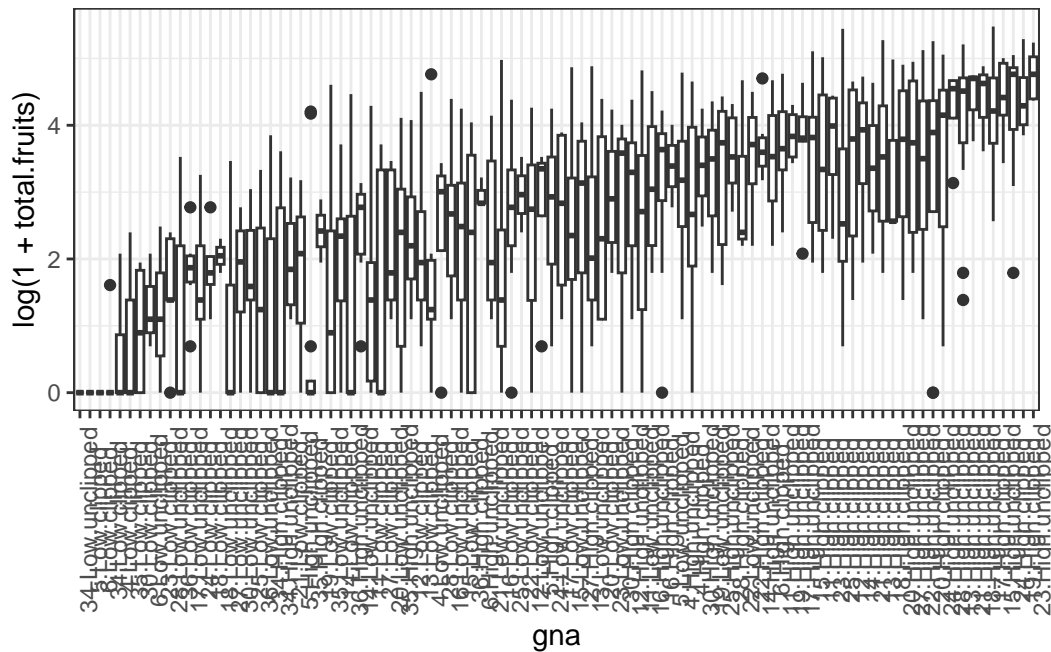


Figure 6.4.: Boxplot of total fruits (log + 1) per genotypes and treatments

We could also calculate the variance for each genotype \times treatment combination and provide a statistical summary of these variances. This reveals substantial variation among the sample variances on the transformed data. In addition to heterogeneous variances across groups, Figure 1 reveals many zeroes in groups, and some groups with a mean and variance of zero, further suggesting we need a non-normal error distribution, and perhaps something other than a Poisson distribution.

We could calculate $\lambda(\text{mean})$ for each genotype \times treatment combination and provide a statistical summary of each group's λ .

```
grp_means <- with(dat_tf, tapply(total.fruits, list(gna), mean))
summary(grp_means)
```

```
Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
0.00  11.35   23.16   31.86  49.74  122.40
```

A core property of the Poisson distribution is that the variance is equal to the mean. A simple diagnostic is a plot of the group variances against the group means:

- Poisson-distributed data will result in a linear pattern with slope = 1
- as long as the variance is generally greater than the mean, we call the data overdispersed. Overdispersion comes in various forms:

- a linear mean-variance relationship with $\text{Var} = \phi\mu$ (a line through the origin) with $\phi > 1$ is called a quasi-Poisson pattern (this term describes the mean-variance relationship, not any particular probability distribution); we can implement it statistically via quasiliikelihood (Venables and Ripley, 2002) or by using a particular parameterization of the negative binomial distribution (“NB1” in the terminology of Hardin and Hilbe (2007))
- a semi-quadratic pattern, $\text{Var} = \mu(1 + \alpha\mu)$ or $\mu(1 + \mu/k)$, is characteristic of overdispersed data that is driven by underlying heterogeneity among samples, either the negative binomial (gamma-Poisson) or the lognormal-Poisson (Elston et al. 2001)

We’ve already calculated the group (genotype \times treatment) means, we calculate the variances in the same way.

```
grp_vars <- with(
  dat_tf,
  tapply(
    total.fruits,
    list(gna), var
  )
)
```

We can get approximate estimates of the quasi-Poisson (linear) and negative binomial (linear/quadratic) pattern using `lm`.

```
lm1 <- lm(grp_vars ~ grp_means - 1) ## `quasi-Poisson' fit
phi_fit <- coef(lm1)
lm2 <- lm((grp_vars - grp_means) ~ I(grp_means^2) - 1)
k_fit <- 1 / coef(lm2)
```

Now we can plot them.

```
plot(grp_vars ~ grp_means, xlab = "group means", ylab = "group variances")
abline(c(0, 1), lty = 2)
text(105, 500, "Poisson")
curve(phi_fit * x, col = 2, add = TRUE)
## bquote() is used to substitute numeric values
## in equations with symbols
text(110, 3900,
  bquote(paste("QP: ", sigma^2 == .(round(phi_fit, 1)) * mu)),
  col = 2
)
curve(x * (1 + x / k_fit), col = 4, add = TRUE)
text(104, 7200, paste("NB: k=", round(k_fit, 1), sep = ""), col = 4)
l_fit <- loess(grp_vars ~ grp_means)
mvec <- 0:120
lines(mvec, predict(l_fit, mvec), col = 5)
text(100, 2500, "loess", col = 5)
```

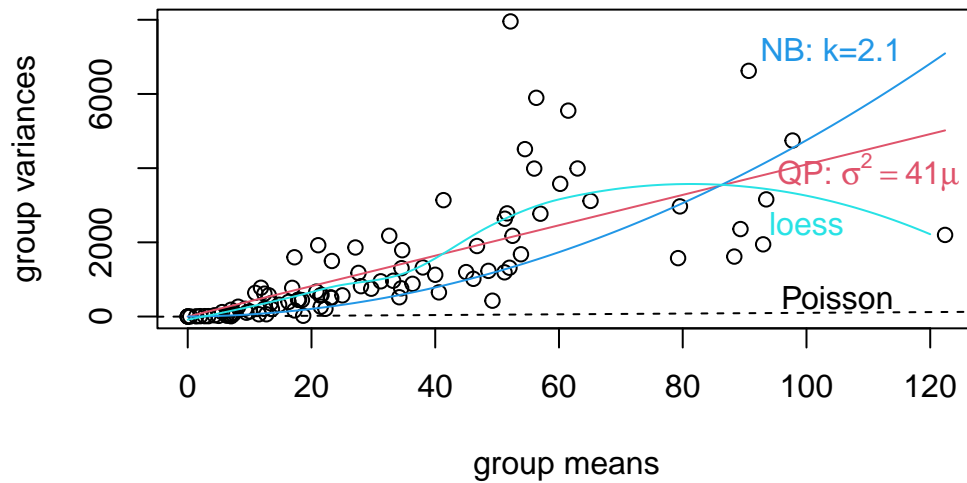


Figure 6.5.: Graphical evaluation of distribution to use

Same with ggplot

```
ggplot(
  data.frame(grp_means, grp_vars),
  aes(x = grp_means, y = grp_vars)) +
  geom_point() +
  geom_smooth(
    aes(colour = "Loess"), se = FALSE) +
  geom_smooth(
    method = "lm", formula = y ~ x - 1, se = FALSE,
    aes(colour = "Q_Pois")) +
  stat_function(
    fun = function(x) x * (1 + x / k_fit),
    aes(colour = "Neg_bin")
  ) +
  geom_abline(
    aes(intercept = 0, slope = 1, colour = "Poisson")) +
  scale_colour_manual(
    name = "legend",
    values = c("blue", "purple", "black", "red")) +
  scale_fill_manual(
    name = "legend",
    values = c("blue", "purple", "black", "red")) +
  guides(fill = FALSE)
```

Warning: The ``scale`` argument of ``guides()`` cannot be ``FALSE``. Use "none" instead as of ggplot2 3.3.4.

`geom_smooth()` using `method = 'loess'` and `formula = 'y ~ x'`

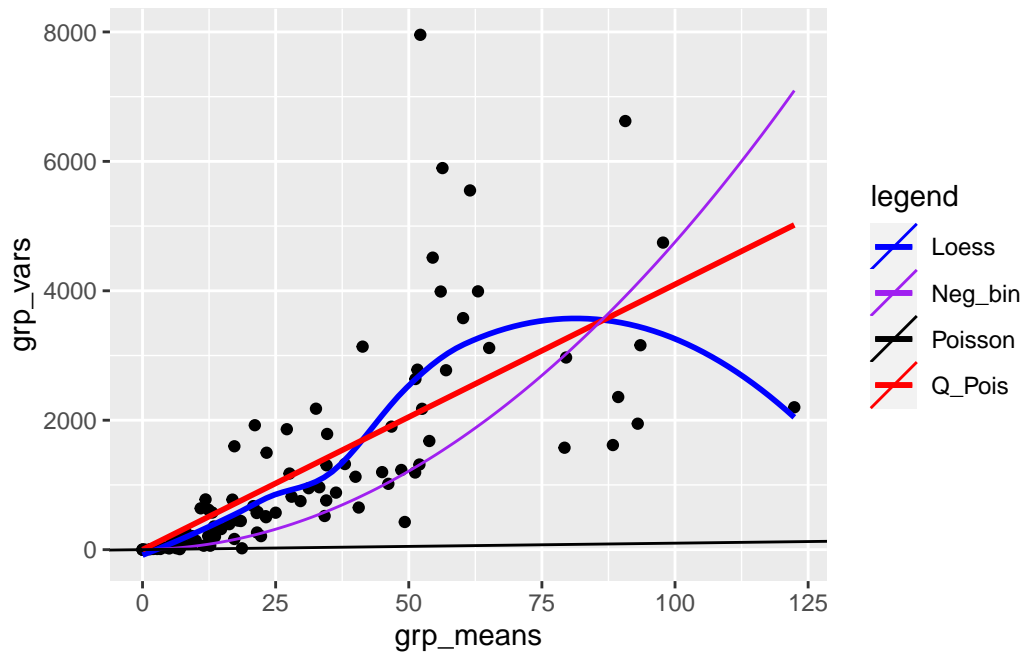



Figure 6.6.: Graphical evaluation of distribution to use with ggplot

These fits are not rigorous statistical tests — they violate a variety of assumptions of linear regression (e.g. constant variance, independence), but they are good enough to give us an initial guess about what distributions we should use.

Exercise

- compare a simple quadratic fit to the data (i.e., without the linear part) with the negative binomial and quasipoisson fits

 Solution


```

lm3 <- lm(grp_vars ~ I(grp_means)^2 - 1) ## quadratic fit
quad_fit <- coef(lm3)

ggplot(
  data.frame(grp_means, grp_vars),
  aes(x = grp_means, y = grp_vars)) +
  geom_point() +
  geom_smooth(
    method = "lm", formula = y ~ x - 1, se = FALSE,
    aes(colour = "Q_Pois")) +
  stat_function(
    fun = function(x) x * (1 + x / k_fit),
    aes(colour = "Neg_bin"))
) +
  geom_smooth(
    method = "lm", formula = y ~ I(x^2) - 1, se = FALSE,
    aes(colour = "Quad")) +
  scale_colour_manual(
    name = "legend",
    values = c("blue", "purple", "black")) +
  scale_fill_manual(
    name = "legend",
    values = c("blue", "purple", "black")) +
  guides(fill = FALSE)

```

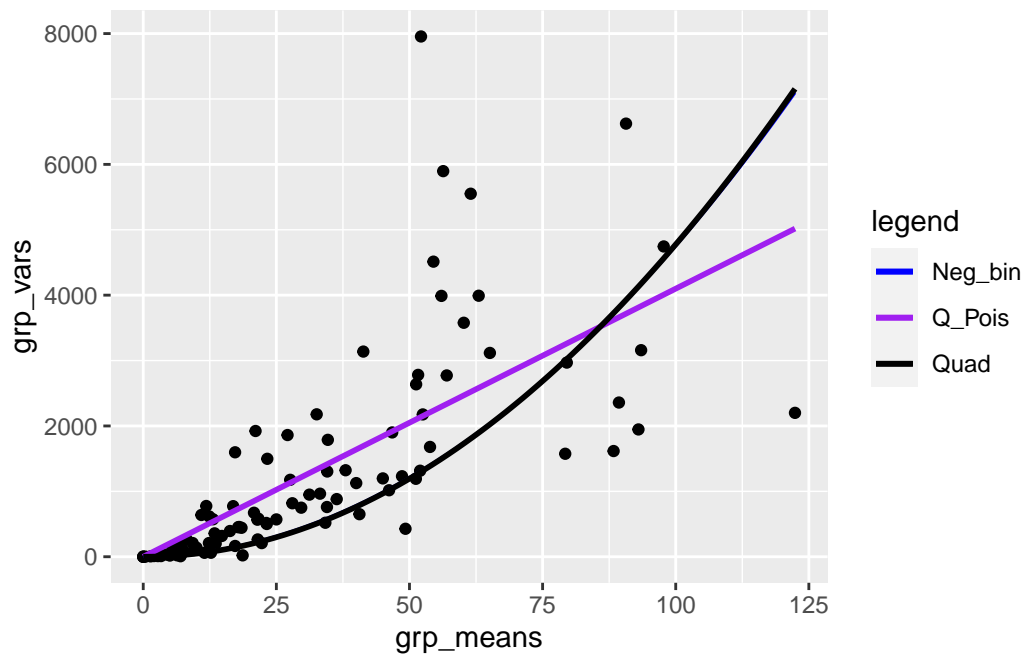


Figure 6.7.: Graphical evaluation of distribution to use including quadratic effect

6.2.5.1. Plotting the response vs treatments

Just to avoid surprises

```
ggplot(dat_tf, aes(x = amd, y = log(total.fruits + 1), colour = nutrient)) +
  geom_point() +
  ## need to use as.numeric(amd) to get lines
  stat_summary(aes(x = as.numeric(amd)), fun = mean, geom = "line") +
  theme_bw() +
  theme(panel.spacing = unit(0, "lines")) +
  facet_wrap(~popu)
```

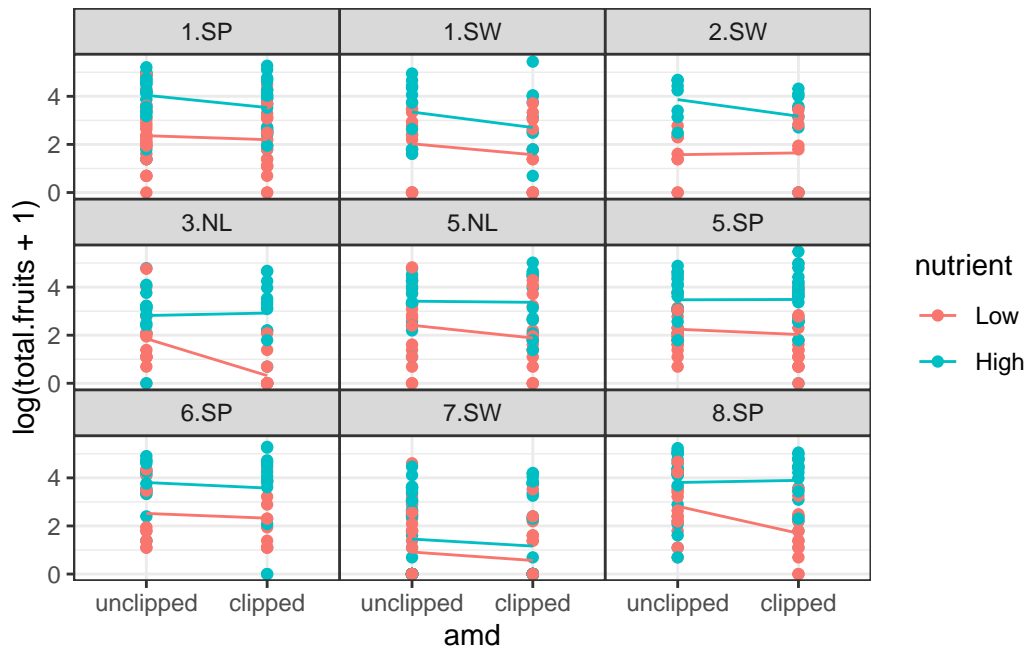


Figure 6.8.: Fruit production by treatments by population

```
ggplot(dat_tf, aes(x = amd, y = log(total.fruits + 1), colour = gen)) +
  geom_point() +
  stat_summary(aes(x = as.numeric(amd)), fun = mean, geom = "line") +
  theme_bw() +
  ## label_both adds variable name ('nutrient') to facet labels
  facet_grid(. ~ nutrient, labeller = label_both)
```

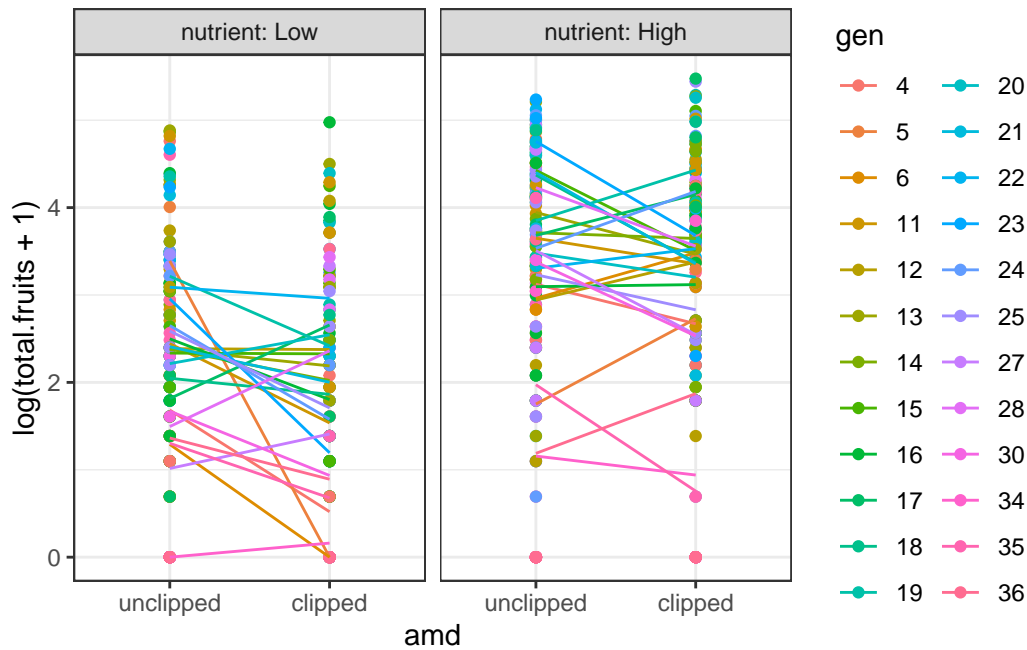


Figure 6.9.: Fruit production by genotype by treatments

6.2.6. Fitting group-wise GLM

Another general starting approach is to fit GLMs to each group of data separately, equivalent to treating the grouping variables as fixed effects. This should result in reasonable variation among treatment effects. We first fit the models, and then examine the coefficients.

```
glm_lis <- lmList(
  total.fruits ~ nutrient * amd | gen,
  data = dat_tf,
  family = "poisson")
plot.lmList(glm_lis)
```

Loading required package: reshape

Attaching package: 'reshape'

The following object is masked from 'package:lubridate':

stamp

The following object is masked from 'package:dplyr':

rename

The following objects are masked from 'package:tidyr':

```
expand, smiths
```

The following object is masked from 'package:Matrix':

```
expand
```

Using `grp` as id variables

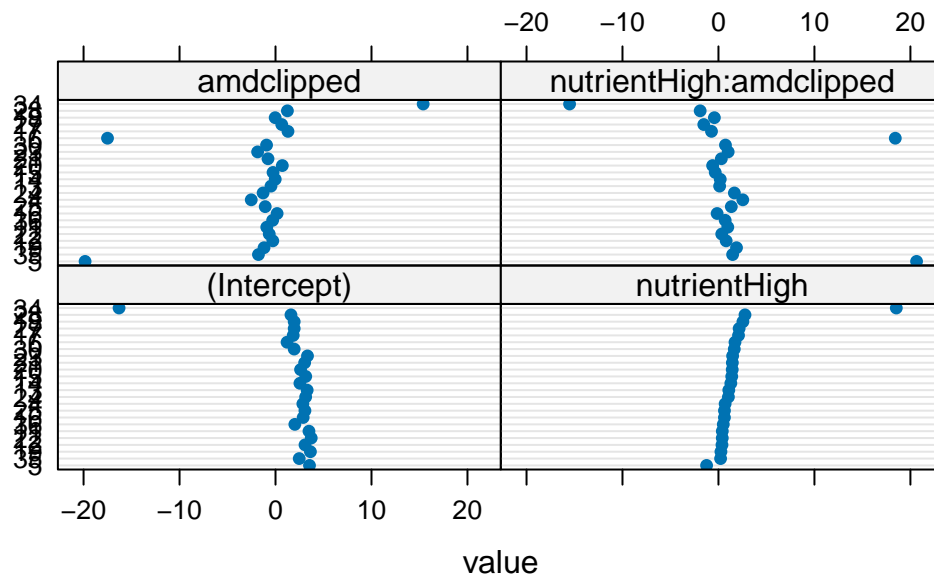


Figure 6.10.: Model coefficients for GLM fits on each genotype

Three genotypes (5, 6, 34) have extreme coefficients (Fig. 5). A mixed model assumes that the underlying random effects are normally distributed, although we shouldn't take these outliers too seriously at this point — we are not actually plotting the random effects, or even estimates of random effects (which are not themselves guaranteed to be normally distributed), but rather separate estimates for each group. Create a plotting function for Q-Q plots of these coefficients to visualize the departure from normality.

```
qqmath.lmList(glm_lis)
```

Using `as` as id variables

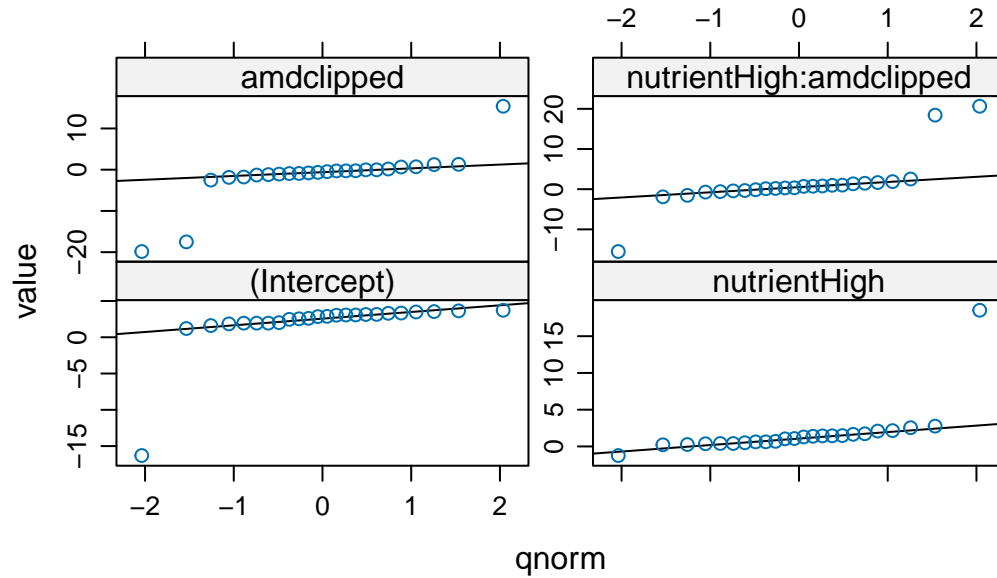


Figure 6.11.: Q-Q plots of model coefficients for GLM fits on each genotype

We see that these extreme coefficients fall far outside a normal error distribution. We shouldn't take these outliers too seriously at this point — we are not actually plotting the random effects, or even estimates of random effects, but rather separate estimates for each group. Especially if these groups have relatively small sample sizes, the estimates may eventually be “shrunk” closer to the mean when we do the mixed model. We should nonetheless take care to see if the coefficients for these genotypes from the GLMM are still outliers, and take the same precautions as we usually do for outliers. For example, we can look back at the original data to see if there is something weird about the way those genotypes were collected, or try re-running the analysis without those genotypes to see if the results are robust.

6.2.7. Fitting and evaluating GLMMs

Now we (try to) build and fit a full model, using `glmer` in the `emoji::emoji("pacakage") lme4`. This model has random effects for all genotype and population \times treatment random effects, and for the nuisance variables for the rack and germination method (status). (Given the mean-variance relationship we saw it's pretty clear that we are going to have to proceed eventually to a model with overdispersion, but we fit the Poisson model first for illustration.)

```
mp1 <- glmer(total.fruits ~ nutrient * amd +
  rack + status +
  (amd * nutrient | popu) +
  (amd * nutrient | gen),
  data = dat_tf, family = "poisson"
)
```


```
Warning in checkConv(attr(opt, "derivs"), opt$par, ctrl = control$checkConv, :
Model failed to converge with max|grad| = 0.0185742 (tol = 0.002, component 1)
```

```
overdisp_fun(mp1)
```

```
      chisq      ratio      p
13909.47140  23.25999  0.00000
```

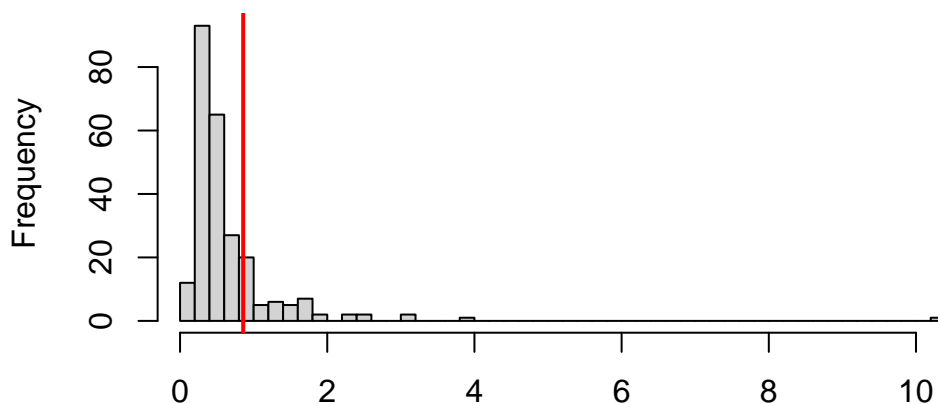
The `overdisp_fun()` is described [here] <https://bbolker.github.io/mixedmodels-misc/glmmFAQ.html#testing-for-overdispersioncomputing-overdispersion-factor>) on the absolutely fantastic FAQ about GLMMs by Ben Bolker <https://bbolker.github.io/mixedmodels-misc/glmmFAQ.html>

We can ignore the model convergence for the moment. This shows that the data are (extremely) over-dispersed, given the model.

We can also use the excellent DHARMA  (Hartig 2022) to evaluate fit of *glm* and *glmm*. So instead of using the function `overdisp_fun()`, we can simply use the function `testDispersion()`.

```
testDispersion(mp1)
```

DHARMA nonparametric dispersion test via sd of residuals fitted vs. simulated



Simulated values, red line = fitted model. p-value (two.sided) = 0.384

```
DHARMA nonparametric dispersion test via sd of residuals fitted vs.
simulated
```

```
data: simulationOutput
dispersion = 1.2934, p-value = 0.384
alternative hypothesis: two.sided
```

As you can see, DHARMA suggests that there is no overdispersion based on the distribution of residuals from simulated data. We are going to consider that we have overdispersion and adjust the model accordingly.

Now we add the observation-level random effect to the model to account for overdispersion (Elston et al. 2001).

```
mp2 <- update(mp1, . ~ . + (1 | X))
```

```
Warning in checkConv(attr(opt, "derivs"), opt$par, ctrl = control$checkConv, :
Model failed to converge with max|grad| = 0.159305 (tol = 0.002, component 1)
```

The model takes much longer to fit (and gives warnings). We look just at the variance components. In particular, if we look at the correlation matrix among the genotype random effects, we see a perfect correlation.

```
attr(VarCorr(mp2)$gen, "correlation")
```

| | (Intercept) | amdclipped | nutrientHigh |
|-------------------------|-------------|------------|--------------|
| (Intercept) | 1.0000000 | -0.9965313 | -0.9877088 |
| amdclipped | -0.9965313 | 1.0000000 | 0.9882474 |
| nutrientHigh | -0.9877088 | 0.9882474 | 1.0000000 |
| amdclipped:nutrientHigh | 0.8321072 | -0.8426404 | -0.9076218 |

| | amdclipped:nutrientHigh |
|-------------------------|-------------------------|
| (Intercept) | 0.8321072 |
| amdclipped | -0.8426404 |
| nutrientHigh | -0.9076218 |
| amdclipped:nutrientHigh | 1.0000000 |

We'll try getting rid of the correlations between clipping (amd) and nutrients, using amd+nutrient instead of amd*nutrient in the random effects specification (here it seems easier to re-do the model rather than using update to add and subtract terms).

```
mp3 <- glmer(total.fruits ~ nutrient * amd +
  rack + status +
  (amd + nutrient | popu) +
  (amd + nutrient | gen) + (1 | X),
  data = dat_tf, family = "poisson"
)
```

```
Warning in checkConv(attr(opt, "derivs"), opt$par, ctrl = control$checkConv, :
Model failed to converge with max|grad| = 0.226429 (tol = 0.002, component 1)
```

```
attr(VarCorr(mp3)$gen, "correlation")
```

```

              (Intercept) amdclipped nutrientHigh
(Intercept)   1.0000000 -0.9981776  -0.9966490
amdclipped    -0.9981776  1.0000000   0.9955458
nutrientHigh  -0.9966490  0.9955458   1.0000000

```

```
attr(VarCorr(mp3)$popu, "correlation")
```

```

              (Intercept) amdclipped nutrientHigh
(Intercept)   1.0000000  0.9970406   0.9974900
amdclipped    0.9970406  1.0000000   0.9937833
nutrientHigh  0.9974900  0.9937833   1.0000000

```

Unfortunately, we still have perfect correlations among the random effects terms. For some models (e.g. random-slope models), it is possible to fit random effects models in such a way that the correlation between the different parameters (intercept and slope in the case of random-slope models) is constrained to be zero, by fitting a model like $(1|f)+(0+x|f)$; unfortunately, because of the way lme4 is set up, this is considerably more difficult with categorical predictors (factors).

We have to reduce the model further in some way in order not to overfit (i.e., in order to not have perfect ± 1 correlations among random effects). It looks like we can't allow both nutrients and clipping in the random effect model at either the population or the genotype level. However, it's hard to know whether we should proceed with amd or nutrient, both, or neither in the model.

A convenient way to proceed if we are going to try fitting several different combinations of random effects is to fit the model with all the fixed effects but only observation-level random effects, and then to use update to add various components to it.

```

mp_obs <- glmer(total.fruits ~ nutrient * amd +
  rack + status +
  (1 | X),
data = dat_tf, family = "poisson"
)


```

Now, for example, `update(mp_obs, . ~ . + (1|gen) + (amd|popu))` fits the model with intercept random effects at the genotype level and variation in clipping effects across populations.

Exercise

Exercise using update, fit the models with

1. clipping variation at both genotype and population levels;
2. nutrient variation at both genotype and populations; convince yourself that trying to fit variation in either clipping or nutrients leads to overfitting (perfect correlations).
3. Fit the model with only intercept variation at the population and genotype levels, saving it as mp4; show that there is non-zero variance estimated

 Solution

1.

```
mpcli <- update(mp_obs, . ~ . + (amd | gen) + (amd | popu))
```

```
Warning in checkConv(attr(opt, "derivs"), opt$par, ctrl = control$checkConv, :
Model failed to converge with max|grad| = 0.0882114 (tol = 0.002, component 1)
```

```
VarCorr(mpcli)
```

| Groups | Name | Std.Dev. | Corr |
|--------|-------------|----------|--------|
| X | (Intercept) | 1.431001 | |
| gen | (Intercept) | 0.296711 | |
| | amdclipped | 0.038708 | -0.887 |
| popu | (Intercept) | 0.754243 | |
| | amdclipped | 0.130903 | 0.997 |

2.

```
mpnut <- update(mp_obs, . ~ . + (nutrient | gen) + (nutrient | popu))
```

```
Warning in checkConv(attr(opt, "derivs"), opt$par, ctrl = control$checkConv, :
Model failed to converge with max|grad| = 0.029394 (tol = 0.002, component 1)
```

```
VarCorr(mpnut)
```

| Groups | Name | Std.Dev. | Corr |
|--------|--------------|----------|--------|
| X | (Intercept) | 1.41977 | |
| gen | (Intercept) | 0.47882 | |
| | nutrientHigh | 0.32631 | -1.000 |
| popu | (Intercept) | 0.74639 | |
| | nutrientHigh | 0.12083 | 1.000 |

3.

```
mp4 <- update(mp_obs, . ~ . + (1 | gen) + (1 | popu))
```


```
VarCorr(mp4)
```

| Groups | Name | Std.Dev. |
|--------|-------------|----------|
| X | (Intercept) | 1.43127 |
| gen | (Intercept) | 0.28582 |
| popu | (Intercept) | 0.80598 |

In other words, while it's biologically plausible that there is some variation in the nutrient or clipping effect at the genotype or population levels, with this modeling approach we really don't have enough data to speak confidently about these effects. Let's check that mp4 no longer incorporates overdispersion (the observation-level random effect should have taken care of it):

```
overdisp_fun(mp4)
```

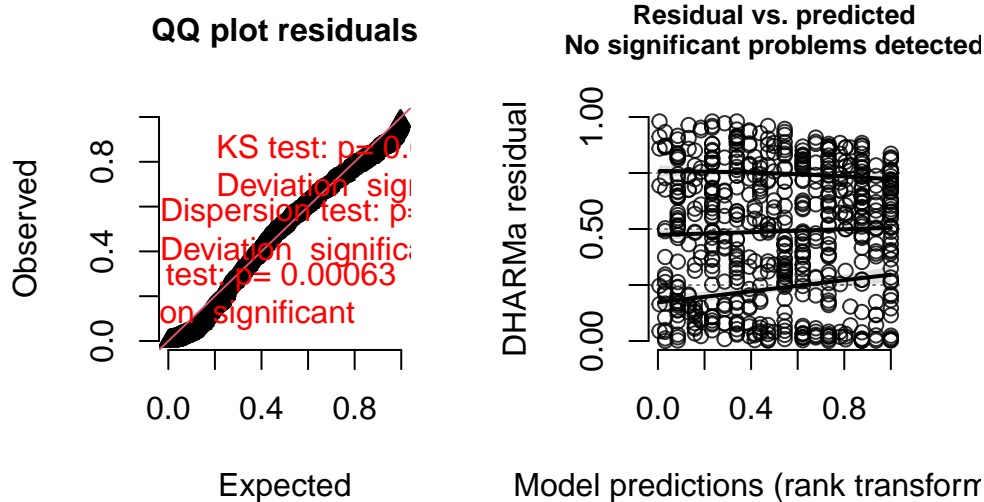
| chisq | ratio | p |
|-------------|-----------|-----------|
| 177.5249980 | 0.2886585 | 1.0000000 |

Using the DHARMA , we will also check the model. To do so we first need to simulate some data and get the *scaled residuals* following the DHARMA notation. Then we can check the distributional properties of the *scaled residuals* and see if they follow the classic assumption using the different functions provided.

```
scaled_res <- simulateResiduals(mp4)
plot(scaled_res)
```

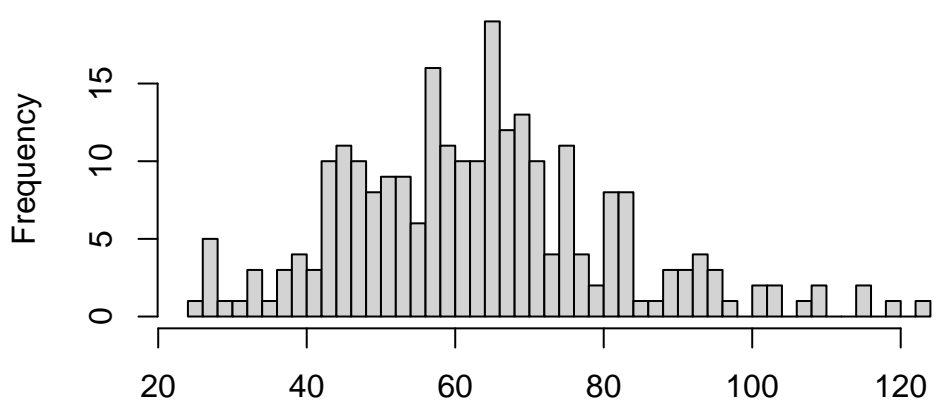
DHARMA: testOutliers with type = binomial may have inflated Type I error rates for integer-valued distributions. To get a more exact result, it is recommended to re-run testOutliers with t

DHARMA residual



```
testZeroInflation(mp4, plot = TRUE)
```

**DHARMA zero-inflation test via comparison to
expected zeros with simulation under H0 = fitted
model**



Simulated values, red line = fitted model. p-value (two.sided) = 0

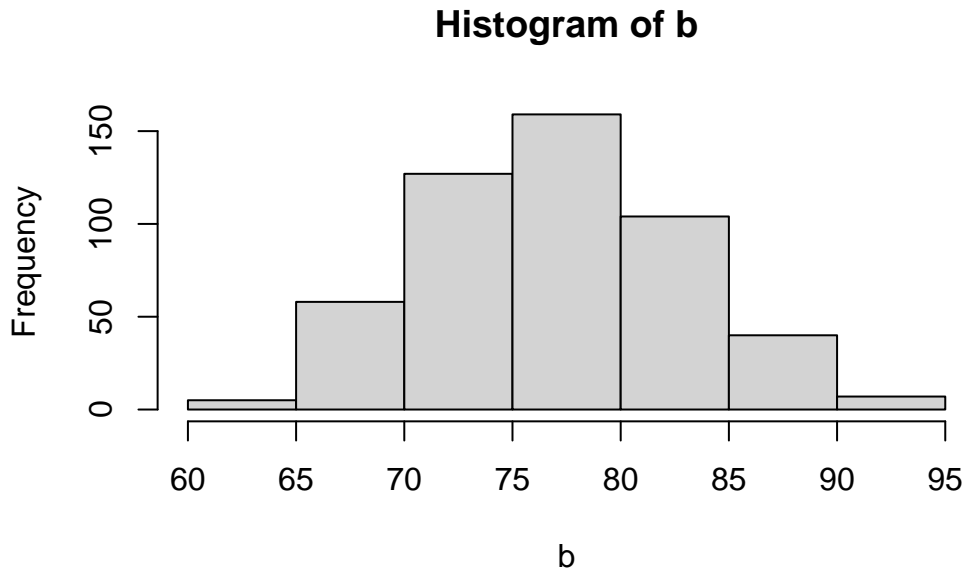
DHARMA zero-inflation test via comparison to expected zeros with
simulation under H0 = fitted model

```
data: simulationOutput
ratioObsSim = 1.9768, p-value < 2.2e-16
alternative hypothesis: two.sided
```

```
# note about overdispersion
sum(dat_tf$total.fruits == 0)
```

```
[1] 126
```

```
a <- predict(mp4, type = "response")
b <- rep(0, 500)
for (j in 1:500) {
  b[j] <- sum(sapply(seq(nrow(dat_tf)), function(i) rpois(1, a[i])) == 0)
}
hist(b)
```



6.2.8. Inference

6.2.8.1. Random effects

`glmer` (`lmer`) does not return information about the standard errors or confidence intervals of the variance components.

```
VarCorr(mp4)
```

| Groups | Name | Std.Dev. |
|--------|-------------|----------|
| X | (Intercept) | 1.43127 |
| gen | (Intercept) | 0.28582 |
| popu | (Intercept) | 0.80598 |

6.2.8.1.1. Testing for random Effects

If we want to test the significance of the random effects we can fit reduced models and run likelihood ratio tests via `anova`, keeping in mind that in this case (testing a null hypothesis of zero variance, where the parameter is on the boundary of its feasible region) the reported *p* value is approximately twice what it should be.

```
mp4v1 <- update(mp_obs, . ~ . + (1 | popu)) ## popu only (drop gen)
mp4v2 <- update(mp_obs, . ~ . + (1 | gen)) ## gen only (drop popu)
```

```
Warning in checkConv(attr("opt", "derivs"), opt$par, ctrl = control$checkConv, :
unable to evaluate scaled gradient
```

```
Warning in checkConv(attr(opt, "derivs"), opt$par, ctrl = control$checkConv, :
Model failed to converge: degenerate Hessian with 2 negative eigenvalues
```

```
anova(mp4, mp4v1)
```

```
Data: dat_tf
```

```
Models:
```

```
mp4v1: total.fruits ~ nutrient + amd + rack + status + (1 | X) + (1 | popu) + nutrient:amd
```

```
mp4: total.fruits ~ nutrient + amd + rack + status + (1 | X) + (1 | gen) + (1 | popu) + nutrient:
```

| | npar | AIC | BIC | logLik | deviance | Chisq | Df | Pr(>Chisq) |
|-------|------|--------|--------|---------|----------|--------|----|------------|
| mp4v1 | 9 | 5017.4 | 5057.4 | -2499.7 | 4999.4 | | | |
| mp4 | 10 | 5015.4 | 5059.8 | -2497.7 | 4995.4 | 4.0639 | 1 | 0.04381 * |

```
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
anova(mp4, mp4v2)
```

```
Data: dat_tf
```

```
Models:
```

```
mp4v2: total.fruits ~ nutrient + amd + rack + status + (1 | X) + (1 | gen) + nutrient:amd
```

```
mp4: total.fruits ~ nutrient + amd + rack + status + (1 | X) + (1 | gen) + (1 | popu) + nutrient:
```

| | npar | AIC | BIC | logLik | deviance | Chisq | Df | Pr(>Chisq) |
|-------|------|--------|--------|---------|----------|--------|----|---------------|
| mp4v2 | 9 | 5031.6 | 5071.5 | -2506.8 | 5013.6 | | | |
| mp4 | 10 | 5015.4 | 5059.8 | -2497.7 | 4995.4 | 18.212 | 1 | 1.976e-05 *** |

```
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

For various forms of linear mixed models, the RLRsim package can do efficient simulation-based hypothesis testing of variance components — unfortunately, that doesn't include GLMMs. If we are sufficiently patient we can do hypothesis testing via brute-force parametric bootstrapping where we repeatedly simulate data from the reduced (null) model, fit both the reduced and full models to the simulated data, and compute the distribution of the deviance (change in $-2 \log$ likelihood). The code below took about half an hour on a reasonably modern desktop computer.

```
simdev <- function() {
  newdat <- simulate(mp4v1)
  reduced <- lme4::refit(mp4v1, newdat)
  full <- lme4::refit(mp4, newdat)
  2 * (c(logLik(full) - logLik(reduced)))
}

set.seed(101)
nulldist0 <- replicate(2, simdev())
## zero spurious (small) negative values
nulldist[nulldist < 0 & abs(nulldist) < 1e-5] <- 0
obsdev <- 2 * c(logLik(mp4) - logLik(mp4v1))
```

```
mean(c(nulldist, obsdev) >= obsdev)
```

```
[1] 0.01492537
```

The true p-value is actually closer to 0.05 than 0.02. In other words, here the deviations from the original statistical model from that for which the original “p value is inflated by 2” rule of thumb was derived — fitting a GLMM instead of a LMM, and using a moderate-sized rather than an arbitrarily large (asymptotic) data set — have made the likelihood ratio test liberal (increased type I error) rather than conservative (decreased type I error).

We can also inspect the random effects estimates themselves (in proper statistical jargon, these might be considered “predictions” rather than “estimates” (Robinson, 1991)). We use the built-in dotplot method for the random effects extracted from glmer fits (i.e. `ranef(model,condVar=TRUE)`), which returns a list of plots, one for each random effect level in the model.

```
r1 <- as.data.frame(ranef(mp4, condVar = TRUE, whichel = c("gen", "popu")))
p1 <- ggplot(subset(r1, grpvar == "gen"), aes(y = grp, x = condval)) +
  geom_point() +
  geom_pointrange(
    aes(xmin = condval - condsd * 1.96, xmax = condval + condsd * 1.96)
  ) +
  geom_vline(aes(xintercept = 0, color = "red")) +
  theme_classic() +
  theme(legend.position = "none")
p2 <- ggplot(subset(r1, grpvar == "popu"), aes(y = grp, x = condval)) +
  geom_point() +
  geom_pointrange(
    aes(xmin = condval - condsd * 1.96, xmax = condval + condsd * 1.96)
  ) +
  geom_vline(aes(xintercept = 0, color = "red")) +
  theme_classic() +
  theme(legend.position = "none")
p1 + p2
```

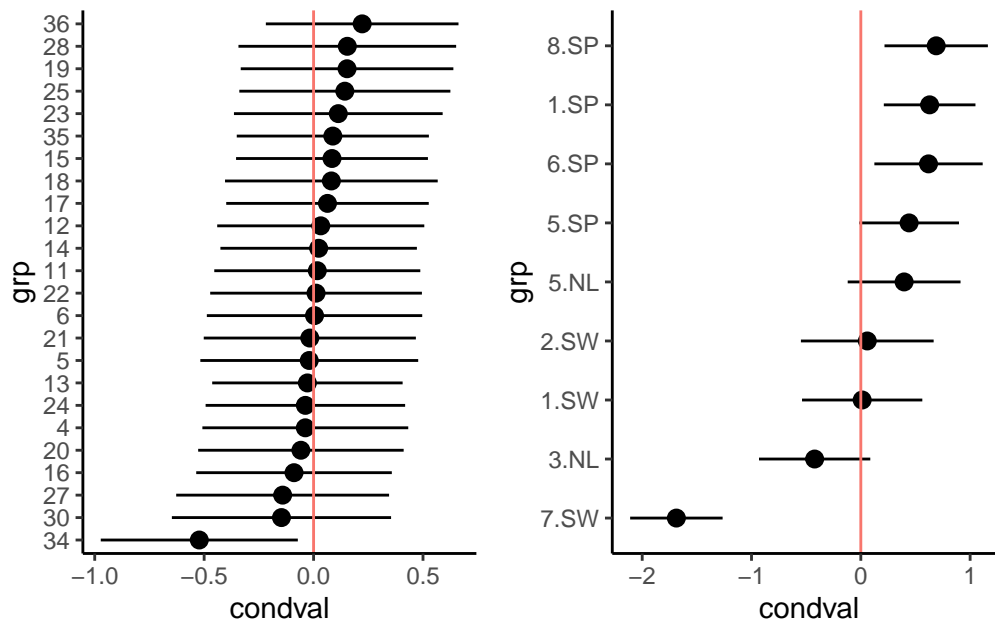


Figure 6.12.: Distribution of BLUPs for genotypes and populations

As expected from the similarity of the variance estimates, the population-level estimates (the only shared component) do not differ much between the two models. There is a hint of regional differentiation — the Spanish populations have higher fruit sets than the Swedish and Dutch populations. Genotype 34 again looks a little bit unusual.

6.2.8.2. Fixed effects

Now we want to do inference on the fixed effects. We use the `drop1` function to assess both the AIC difference and the likelihood ratio test between models. (In `glmm_funs.R` we define a convenience function `dfun` to convert the AIC tables returned by `drop1` (which we will create momentarily) into AIC tables.) Although the likelihood ratio test (and the AIC) are asymptotic tests, comparing fits between full and reduced models is still more accurate than the Wald (curvature-based) tests shown in the summary tables for `glmer` fits.

```
(dd_aic <- dfun(drop1(mp4)))
```

```
Warning in checkConv(attr(opt, "derivs"), opt$par, ctrl = control$checkConv, :
Model failed to converge with max|grad| = 0.00959403 (tol = 0.002, component 1)
```

Single term deletions

Model:

```
total.fruits ~ nutrient + amd + rack + status + (1 | X) + (1 |
  gen) + (1 | popu) + nutrient:amd
```

| | npar | dAIC |
|--------|------|--------|
| <none> | | 0.000 |
| rack | 1 | 55.083 |

```
status          2  1.612
nutrient:amd    1  1.444
```

```
(dd_lrt <- drop1(mp4, test = "Chisq"))
```

```
Warning in checkConv(attr(opt, "derivs"), opt$par, ctrl = control$checkConv, :
Model failed to converge with max|grad| = 0.00959403 (tol = 0.002, component 1)
```

Single term deletions

Model:

```
total.fruits ~ nutrient + amd + rack + status + (1 | X) + (1 |
  gen) + (1 | popu) + nutrient:amd
```

| | npar | AIC | LRT | Pr(Chi) | |
|--------------|------|--------|--------|-----------|-----|
| <none> | | 5015.4 | | | |
| rack | 1 | 5070.5 | 57.083 | 4.179e-14 | *** |
| status | 2 | 5017.0 | 5.612 | 0.06044 | . |
| nutrient:amd | 1 | 5016.8 | 3.444 | 0.06349 | . |

```
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

On the basis of these comparisons, there appears to be a very strong effect of rack and weak effects of status and of the interaction term. Dropping the nutrient:amd interaction gives a (slightly) increased AIC (AIC = 1.4), so the full model has the best expected predictive capability (by a small margin). On the other hand, the p-value is slightly above 0.05 ($p = 0.06$). At this point we remove the non-significant interaction term so we can test the main effects. (We don't worry about removing status because it measures an aspect of experimental design that we want to leave in the model whether it is significant or not.) Once we have fitted the reduced model, we can run the LRT via anova.

```
mp5 <- update(mp4, . ~ . - amd:nutrient)
```

```
Warning in checkConv(attr(opt, "derivs"), opt$par, ctrl = control$checkConv, :
Model failed to converge with max|grad| = 0.00959403 (tol = 0.002, component 1)
```

```
anova(mp5, mp4)
```

Data: dat_tf

Models:

```
mp5: total.fruits ~ nutrient + amd + rack + status + (1 | X) + (1 | gen) + (1 | popu)
```

```
mp4: total.fruits ~ nutrient + amd + rack + status + (1 | X) + (1 | gen) + (1 | popu) + nutrient:
```

| | npar | AIC | BIC | logLik | deviance | Chisq | Df | Pr(>Chisq) |
|-----|------|--------|--------|---------|----------|--------|----|------------|
| mp5 | 9 | 5016.8 | 5056.8 | -2499.4 | 4998.8 | | | |
| mp4 | 10 | 5015.4 | 5059.8 | -2497.7 | 4995.4 | 3.4439 | 1 | 0.06349 . |

```
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```


Exercise Test now the reduced model.

In the reduced model, we find that both nutrients and clipping have strong effects, whether measured by AIC or LRT. If we wanted to be still more careful about our interpretation, we would try to relax the asymptotic assumption. In classical linear models, we would do this by doing F tests with the appropriate denominator degrees of freedom. In “modern” mixed model approaches, we might try to use denominator-degree-of-freedom approximations such as the Kenward-Roger (despite the controversy over these approximations, they are actually available in `lmerTest`, but they do not apply to GLMMs. We can use a parametric bootstrap comparison between nested models to test fixed effects, as we did above for random effects, with the caveat that is computationally slow.

In addition, we can check the normality of the random effects and find they are reasonable (Fig. 10).

```
r5 <- as.data.frame(ranef(mp5))
ggplot(data = r5, aes(sample = condval)) +
  geom_qq() + geom_qq_line() +
  facet_wrap(~ grpvar) +
  theme_classic()
```

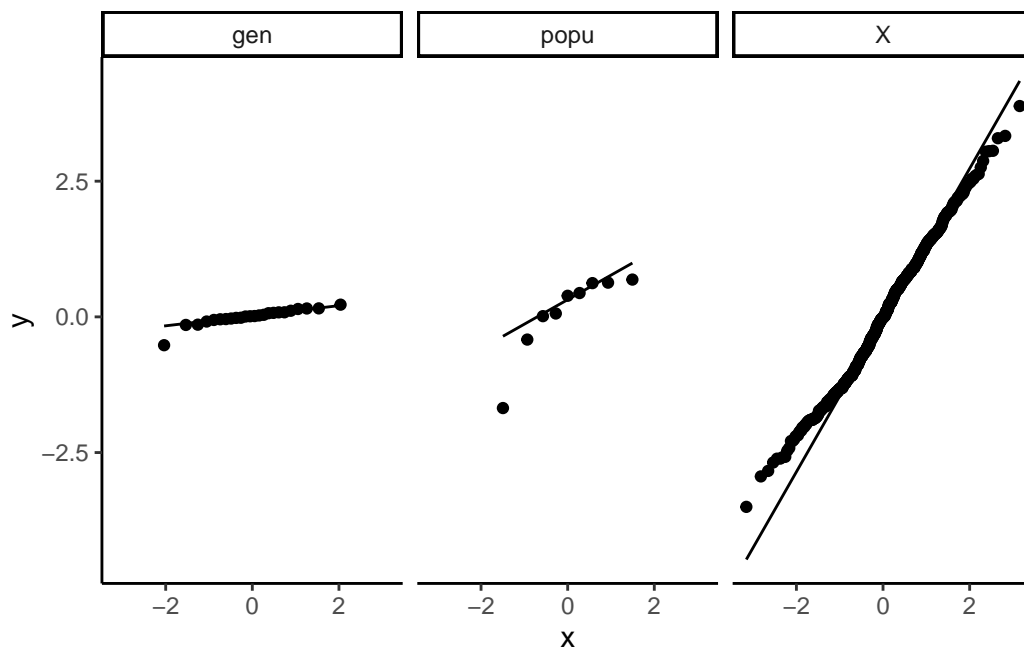
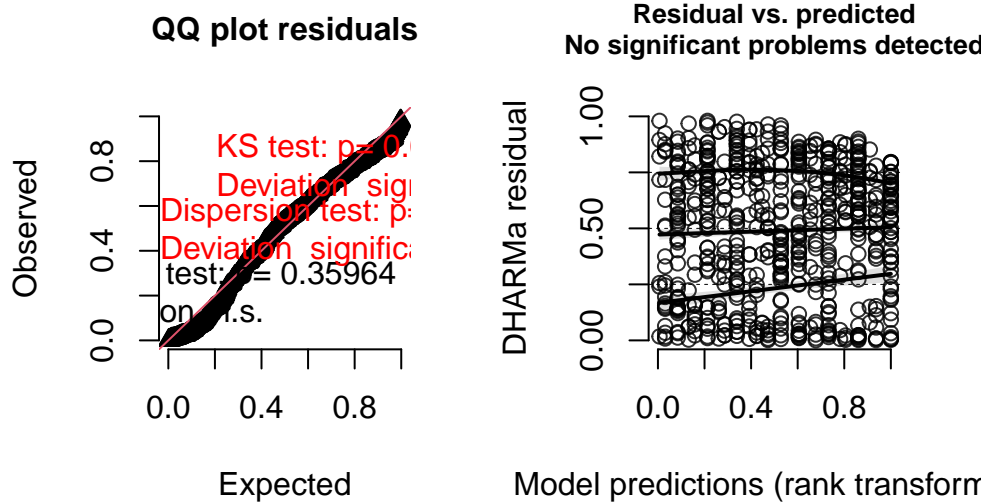


Figure 6.13.: Q-Q plot of BLUPs from model mp5

Checking everything with DHARMA also

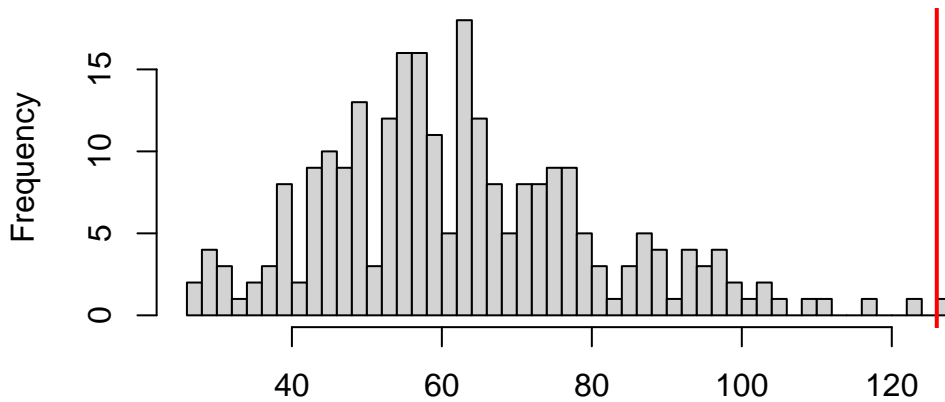
```
scaled_res <- simulateResiduals(mp5)
plot(scaled_res)
```

DHARMA residual



```
testZeroInflation(mp5, plot = TRUE)
```

DHARMA zero-inflation test via comparison to expected zeros with simulation under H0 = fitted model



Simulated values, red line = fitted model. p-value (two.sided) = 0.008

DHARMA zero-inflation test via comparison to expected zeros with simulation under H0 = fitted model

```
data: simulationOutput
ratioObsSim = 1.9883, p-value = 0.008
alternative hypothesis: two.sided
```

It is better than before but not perfect. I think this is completely OK and that it will extremely rarely be perfect. You need to learn what is acceptable (by that I mean you find acceptable) and be happy to justify and discuss your decisions.

6.2.9. Conclusions

Our final model includes fixed effects of nutrients and clipping, as well as the nuisance variables rack and status; observation-level random effects to account for overdispersion; and variation in overall fruit set at the population and genotype levels. However, we don't (apparently) have quite enough information to estimate the variation in clipping and nutrient effects, or their interaction, at the genotype or population levels. There is a strong overall positive effect of nutrients and a slightly weaker negative effect of clipping. The interaction between clipping and nutrients is only weakly supported (i.e. the p-value is not very small), but it is positive and about the same magnitude as the clipping effect, which is consistent with the statement that "nutrients cancel out the effect of herbivory".

Exercise

Exercise

- Re-do the analysis with region as a fixed effect.
- Re-do the analysis with a one-way layout as suggested above

6.2.10. Happy generalized mixed-modelling



Figure 6.14.: A GLMM character

7. Introduction to Bayesian Inference

7.1. Lecture

Amazing beasties and crazy animals



Figure 7.1.: Dream pet dragon

7.1.1. Bayes' theorem

First, let's review the theorem. Mathematically, it says how to convert one conditional probability into another one.

$$P(B | A) = \frac{P(A | B) * P(B)}{P(A)}$$

The formula becomes more interesting in the context of statistical modeling. We have some model that describes a data-generating process and we have some *observed* data, but we want to estimate some *unknown* model parameters. In that case, the formula reads like:

$$P(\text{hypothesis} | \text{data}) = \frac{P(\text{data} | \text{hypothesis}) * P(\text{hypothesis})}{P(\text{data})}$$

These terms have conventional names:

$$\text{posterior} = \frac{\text{likelihood} * \text{prior}}{\text{evidence}}$$

Prior and *posterior* describe when information is obtained: what we know pre-data is our prior information, and what we learn post-data is the updated information (“posterior”).

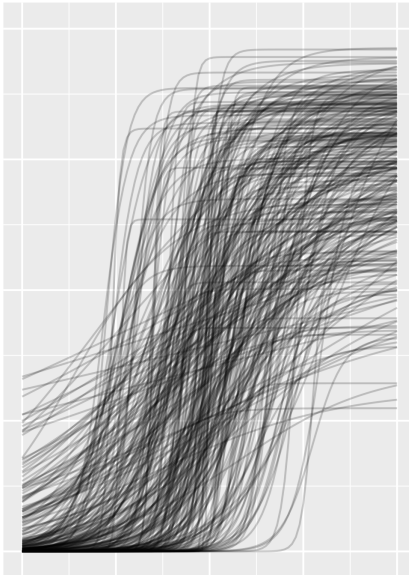
The *likelihood* in the equation says how likely the data is given the model parameters. I think of it as *fit*: How well do the parameters fit the data? Classical regression’s line of best fit is the maximum likelihood line. The likelihood also encompasses the data-generating process behind the model. For example, if we assume that the observed data is normally distributed, then we evaluate the likelihood by using the normal probability density function. You don’t need to know what that last sentence means. What’s important is that the likelihood contains our built-in assumptions about how the data is distributed.

The *evidence* (sometimes called *average likelihood*) is harder to grasp. I am not sure how to describe it in an intuitive way. It’s there to make sure the math works out so that the posterior probabilities sum to 1. Some presentations of Bayes’ theorem gloss over it and I am not the exception 😊. The important thing to note is that the posterior is proportional to the likelihood and prior information.

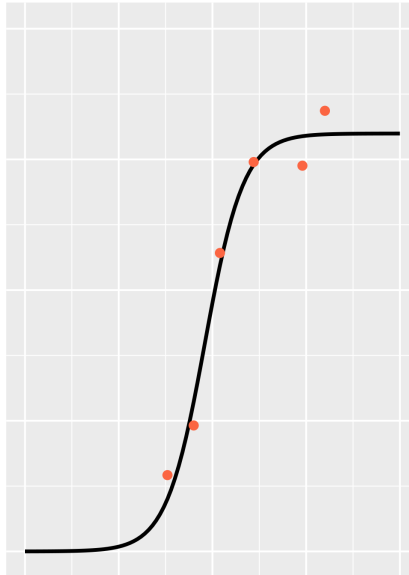
$$\text{posterior information} \propto \text{likelihood of data} * \text{prior information}$$

So simply put, **you update your prior information in proportion to how well it fits the observed data**. So essentially you are doing that on a daily basis for everything except when you are doing frequentist stats 😊.

Plausible curves before seeing data



How well do the curves fit the data



Plausible curves after seeing data

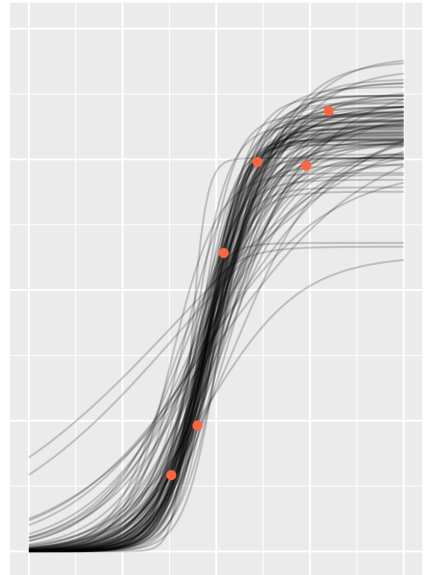


Figure 7.2.: Bayesian Triptych

 Warning

A word of encouragement! The prior is an intimidating part of Bayesian statistics. It seems highly subjective, as though we are pulling numbers from thin air, and it can be overwhelming for complex models. But if we are familiar with the kind of data we are modeling, we have prior information. We can have the model simulate new observations using the prior distribution and then plot the hypothetical data. Does anything look wrong or implausible about the simulated data? If so, then we have some prior information that we can include in our model. Note that we do not evaluate the plausibility of the simulated data based on the data we have in hand (the data we want to model); that’s not

7.1.2. Intro to MCMC

We will now walk through a simple example coded in R to illustrate how an MCMC algorithm works.

Suppose you are interested in the mean heart rate is of students when asked a question in a stat course. You are not sure what the exact mean value is, but you know the values are normally distributed with a standard deviation of 15. You have observed 5 individuals to have heart rate of 104, 120, 160, 90, 130. You could use MCMC sampling to draw samples from the target distribution. We need to specify:

1. the starting value for the chain.
2. the length of the chain. In general, more iterations will give you more accurate output.

```
set.seed(170)
hr_obs <- c(104, 112, 132, 115, 110)

start_value <- 250

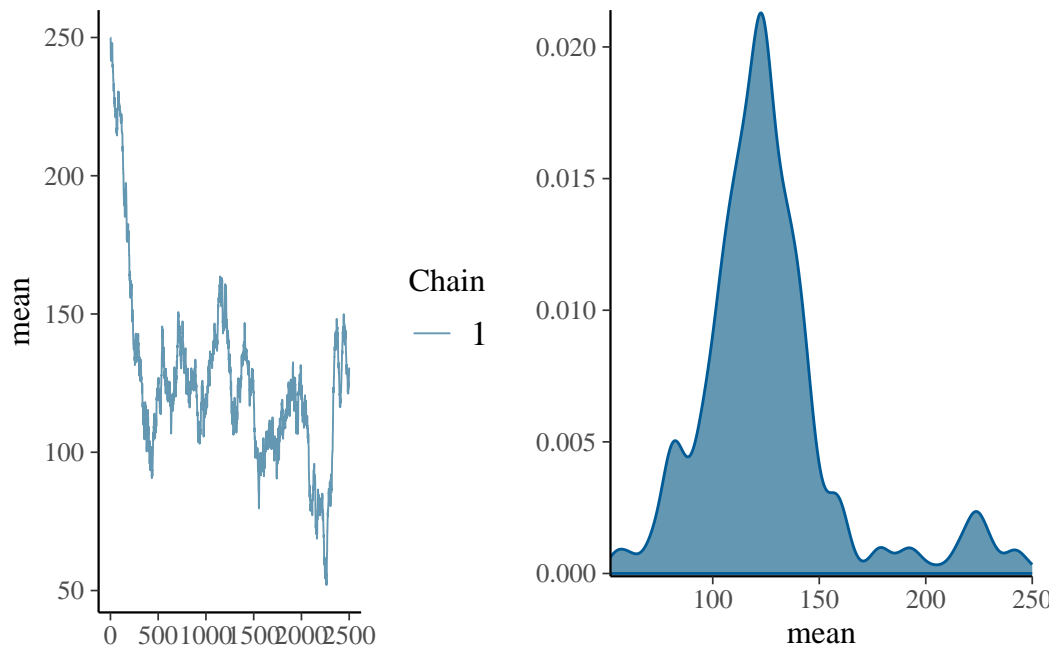
n_iter <- 2500 # define number of iterations

pd_mean <- numeric(n_iter) # create vector for sample values

pd_mean[1] <- start_value # define starting value

for (i in 2:n_iter) {
  proposal <- pd_mean[i - 1] + MASS::mvrnorm(1, 0, 5) # proposal
  lprop <- sum(dnorm(proposal, hr_obs, 15)) # likelihood of proposed parameter
  lprev <- sum(dnorm(pd_mean[i - 1], hr_obs, 15))
  if (lprop / lprev > runif(1)) { # if likelihood of proposed > likelihood previous accept
    # and if likelihood is lower accept with random noise
    pd_mean[i] <- proposal
  } # if true sample the proposal
  else {
    (pd_mean[i] <- pd_mean[i - 1])
  } # if false sample the current value
}

pd_mean <- as.mcmc(data.frame(mean = pd_mean))
mcmc_combo(pd_mean, combo = c("trace", "dens"))
```



```
summary(pd_mean)
```

```
Iterations = 1:2500
Thinning interval = 1
Number of chains = 1
Sample size per chain = 2500
```

1. Empirical mean and standard deviation for each variable, plus standard error of the mean:

| Mean | SD | Naive SE | Time-series SE |
|----------|---------|----------|----------------|
| 125.8105 | 32.8672 | 0.6573 | 13.3046 |

2. Quantiles for each variable:

| 2.5% | 25% | 50% | 75% | 97.5% |
|-------|--------|--------|--------|--------|
| 75.53 | 108.03 | 122.19 | 136.12 | 225.46 |

```
set.seed(170)
hr_obs <- c(104, 112, 132, 115, 110)
n_iter <- 2500 # define number of iterations

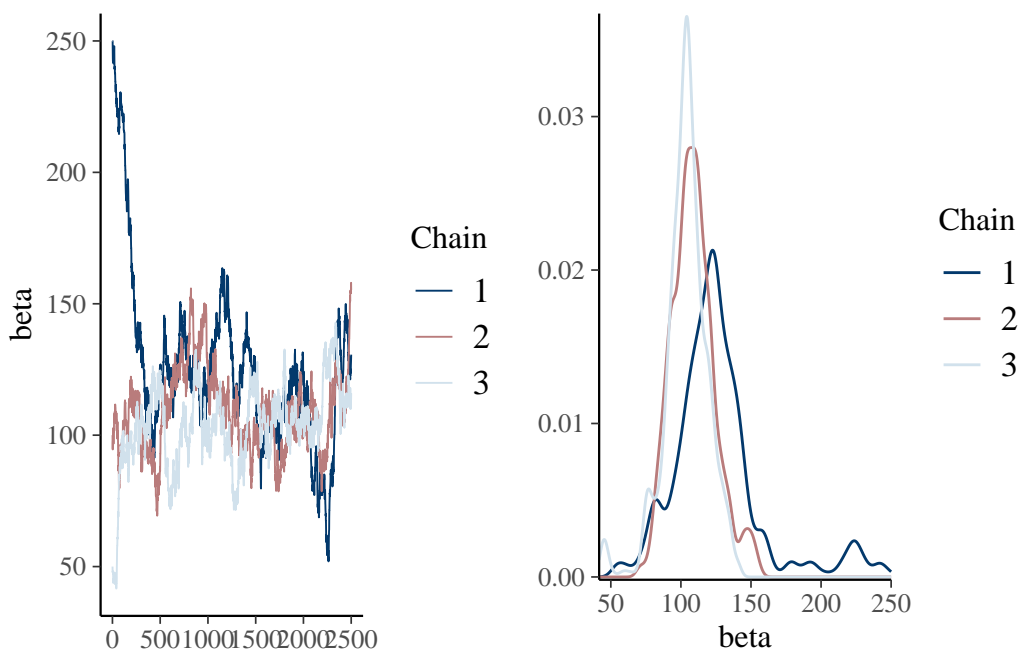
n_chain <- 3
start_value <- c(250, 100, 50)
```



```

pd_mean <- array(NA, dim = c(n_iter, n_chain, 1), dimnames = list(iter = NULL, chain = NULL, para
for (j in seq_len(n_chain)) {
  pd_mean[1, j, 1] <- start_value[j] # define starting value
  for (i in 2:n_iter) {
    proposal <- pd_mean[i - 1, j, 1] + MASS::mvrnorm(1, 0, 5) # proposal
    if (sum(dnorm(proposal, hr_obs, 15)) # likelihood of proposed parameter
        / sum(dnorm(pd_mean[i - 1, j, 1], hr_obs, 15)) > runif(1, 0, 1)) {
      pd_mean[i, j, 1] <- proposal
    } # if true sample the proposal
    else {
      (pd_mean[i, j, 1] <- pd_mean[i - 1, j, 1])
    } # if false sample the current value
  }
}
color_scheme_set("mix-blue-red")
mcmc_combo(pd_mean, combo = c("trace", "dens_overlay"))

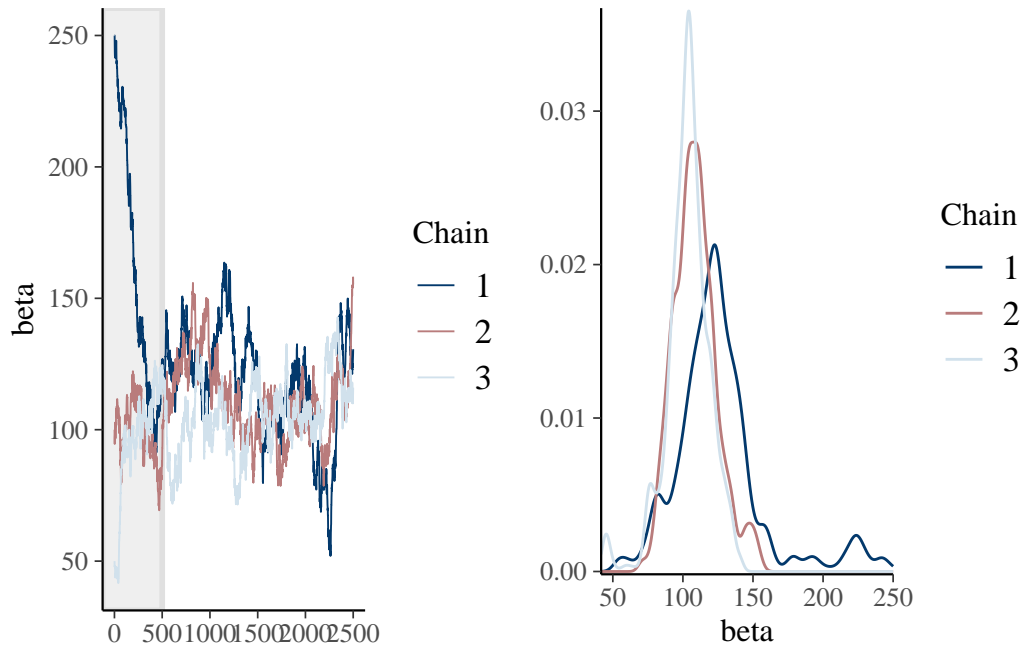
```



```
summary(pd_mean)
```

| Min. | 1st Qu. | Median | Mean | 3rd Qu. | Max. |
|-------|---------|--------|--------|---------|--------|
| 41.65 | 99.32 | 109.68 | 112.71 | 122.52 | 250.00 |

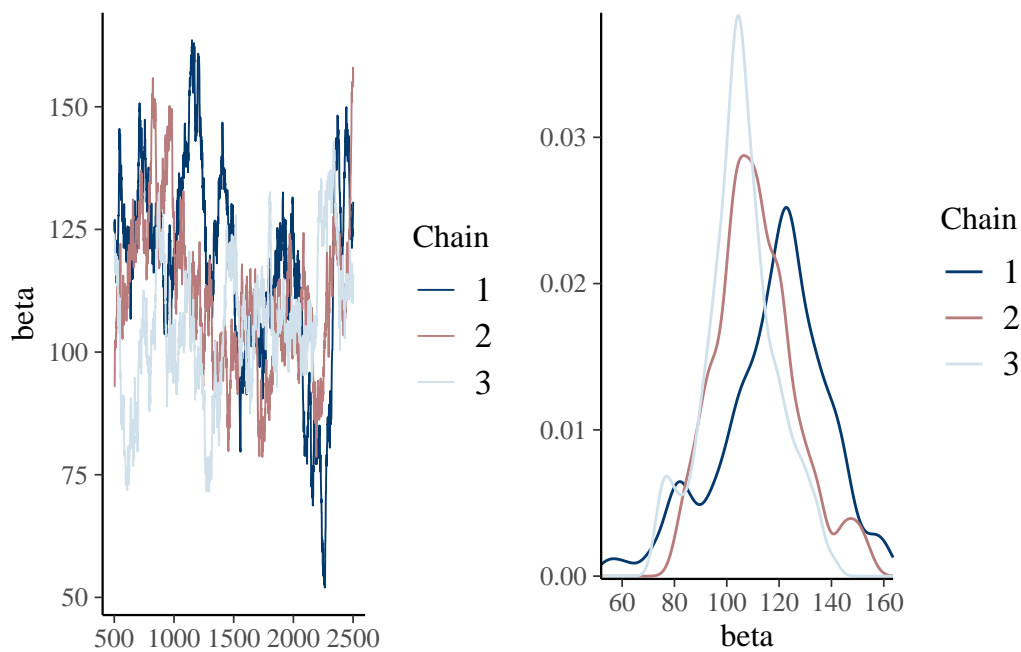
```
mcmc_combo(pd_mean, combo = c("trace", "dens_overlay"), n_warmup = 500)
```



```
pd_burn <- pd_mean[-c(1:500), , , drop = FALSE]
summary(pd_burn)
```

| Min. | 1st Qu. | Median | Mean | 3rd Qu. | Max. |
|-------|---------|--------|--------|---------|--------|
| 51.98 | 100.71 | 110.38 | 111.42 | 122.69 | 163.58 |

```
mcmc_combo(pd_burn, combo = c("trace", "dens_overlay"), iter1 = 501)
```



7.1.3. Inferences

7.1.3.1. Fixed effects

Easy peazy lemon squeezy just have a look at the posteriro distribution, does it overlap 0 yes or no.

talk about mean, median and mode of a distribution as well as credible intervals

7.1.3.2. Random effects

Quite a bit more harder. because constrained to be positive

- Interpreting posterior distribution
- DIC
- WAIC

7.2. Practical

In this practical, we will revisit our analysis on unicorn aggressivity. Honestly, we can use any other data with repeated measures for this exercise but I just love unicorns ❤️. However, instead of fitting the model using `lmer()` from the `lmerTest` 📦 (Kuznetsova et al. 2017), we will refit the model using 2 excellent softwares fitting models with a Bayesian approach: `MCMCglmm` (Hadfield 2010) and `brms` (Bürkner 2021).

7.2.1. R packages needed

First we load required libraries

```
library(lmerTest)
library(tidyverse)
library(rptR)
library(brms)
library(MCMCglmm)
library(bayesplot)
```

7.2.2. A refresher on unicorn ecology

The last model on unicorns was:

```
aggression ~ opp_size + scale(body_size, center = TRUE, scale = TRUE)
  + scale(assay_rep, scale = FALSE) + block
  + (1 | ID)
```

Those scaled terms are abit a sore for my eyes and way too long if we need to type them multiple times in this practical. So first let's recode them. -

```
unicorns <- read.csv("data/unicorns_aggression.csv")
unicorns <- unicorns %>%
  mutate(
    body_size_sc = scale(body_size),
    assay_rep_sc = scale(assay_rep, scale = FALSE)
  )
```

Ok now we can fit the same model by just using:

```
aggression ~ opp_size + body_size_sc + assay_rep_sc + block
+ (1 | ID)
```

We can now fit a model using `lmer()`. Since we want to compare a bit REML and Bayesian aproaches, I am going to wrap the model function in a function called `system.time()`. This function simply estimate the user and computer time use by the function.

```
mer_time <- system.time(
  m_mer <- lmer(
    aggression ~ opp_size + body_size_sc + assay_rep_sc + block
    + (1 | ID),
    data = unicorns
  )
)
mer_time
```

```
   user  system elapsed
0.116   0.000   0.119
```

```
summary(m_mer)
```

```
Linear mixed model fit by REML. t-tests use Satterthwaite's method [
lmerModLmerTest]
Formula: aggression ~ opp_size + body_size_sc + assay_rep_sc + block +
(1 | ID)
Data: unicorns
```

```
REML criterion at convergence: 1136.5
```

```
Scaled residuals:
```

```

      Min       1Q   Median       3Q      Max
-2.85473 -0.62831  0.02545  0.68998  2.74064

```

Random effects:

```

Groups   Name             Variance Std.Dev.
ID       (Intercept)  0.02538  0.1593
Residual                    0.58048  0.7619

```

Number of obs: 480, groups: ID, 80

Fixed effects:

```

              Estimate Std. Error      df t value Pr(>|t|)
(Intercept)   9.00181    0.03907  78.07315  230.395 <2e-16 ***
opp_size      1.05141    0.04281  396.99857   24.562 <2e-16 ***
body_size_sc  0.03310    0.03896  84.21144    0.850  0.398
assay_rep_sc -0.05783    0.04281  396.99857   -1.351  0.177
block        -0.02166    0.06955  397.00209   -0.311  0.756
---

```

Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Correlation of Fixed Effects:

```

              (Intr) opp_sz bdy_s_ assy__
opp_size      0.000
body_siz_sc   0.000  0.000
assay_rp_sc   0.000 -0.100  0.000
block         0.000  0.000  0.002  0.000

```

Ok so it took no time at all to do it and we got our “classic” results.

7.2.3. MCMCglmm

What makes `MCMCglmm` so useful and powerful 🍌 in ecology and for *practical Bayesian people* is that:

1. it is blazing fast 🚀 (for Bayesian analysis) for some models particularly models with structured covariances
2. it is fairly intuitive to code

but it also has some inconvenients:

1. it is blazing fast for **Bayesian analysis** meaning it is 🍌 compared to *maximum likelihood* approaches
2. it has some limitations in terms of functionality, distribution availability and model specifications compared to other *Bayesian* softwares
3. the priors, *oh, the priors* 😭, are a bit tricky to code and understand 🤖.

7.2.3.1. Fitting the Model

So here is how we can code the model in `MCMCglmm()`. It is fairly similar to `lmer()` except that the random effects are specified in a different *argument*.

```
mcglm_time <- system.time(  
  m_mcmcglmm <- MCMCglmm(  
    aggression ~ opp_size + body_size_sc + assay_rep_sc + block,  
    random = ~ID,  
    data = unicorns  
  )  
)
```

```
MCMC iteration = 0  
MCMC iteration = 1000  
MCMC iteration = 2000  
MCMC iteration = 3000  
MCMC iteration = 4000  
MCMC iteration = 5000  
MCMC iteration = 6000  
MCMC iteration = 7000  
MCMC iteration = 8000  
MCMC iteration = 9000  
MCMC iteration = 10000  
MCMC iteration = 11000  
MCMC iteration = 12000  
MCMC iteration = 13000
```

```
summary(m_mcmcglmm)
```

```
Iterations = 3001:12991  
Thinning interval = 10  
Sample size = 1000
```

```
DIC: 1128.004
```

```

G-structure: ~ID

      post.mean 1-95% CI u-95% CI eff.samp
ID 0.003686 9.807e-14 0.0262 45.81

R-structure: ~units

      post.mean 1-95% CI u-95% CI eff.samp
units 0.6044 0.5228 0.6819 1000

Location effects: aggression ~ opp_size + body_size_sc + assay_rep_sc + block

      post.mean 1-95% CI u-95% CI eff.samp pMCMC
(Intercept) 9.00152 8.93150 9.07158 1000 <0.001 ***
opp_size 1.04940 0.96813 1.12946 1000 <0.001 ***
body_size_sc 0.03154 -0.03985 0.09563 1000 0.410
assay_rep_sc -0.05620 -0.13196 0.03546 893 0.184
block -0.02069 -0.16186 0.11553 1000 0.774
---
Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

```

```
mcglm_time
```

```

user system elapsed
1.928 0.000 1.930

```

Model is slow and not good. We need more iteration and maybe even a longer burnin, and honestly maybe better priors.

We can still take the time to have a look at the R object output from `MCMCglmm()`. The 2 main parts we are interested in are:

- `Sol` which stand for the model solution and includes the posteriro distribution of the fixed effects
- `VCV`, for the variance covariance estimates, which includes the posterior distribution of all (co)variances estimates for both random effects and residual variance.

```

omar <- par()
par(mar = c(4, 2, 1.5, 2))
plot(m_mcmcglmm$Sol)

```

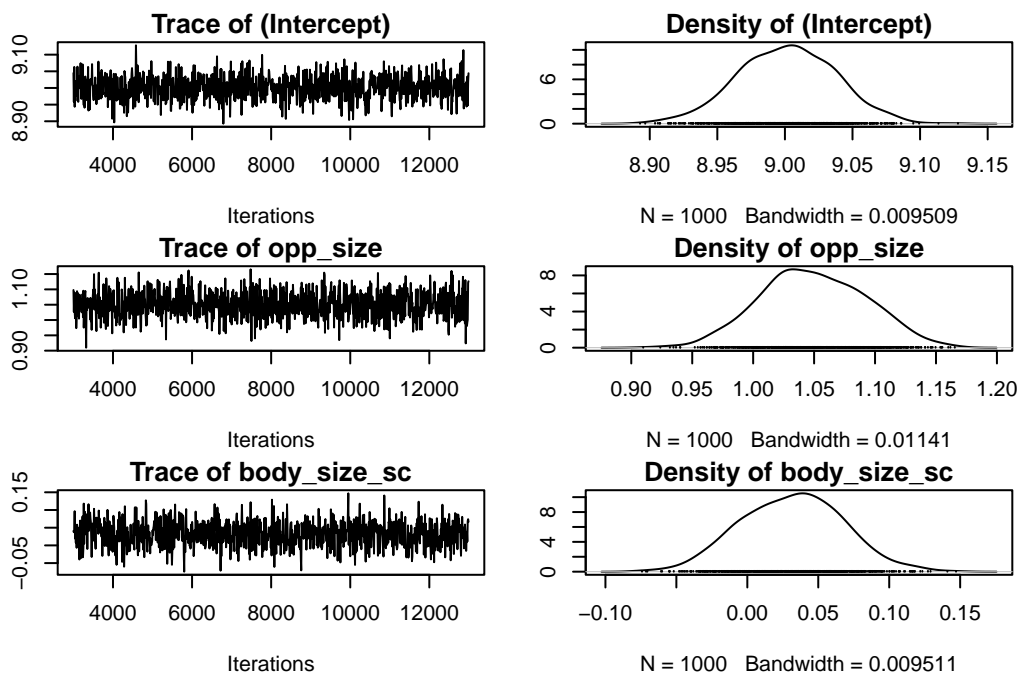


Figure 7.3.: Posterior trace and distribution of the parameters in `m_mcmcglmm` using default settings

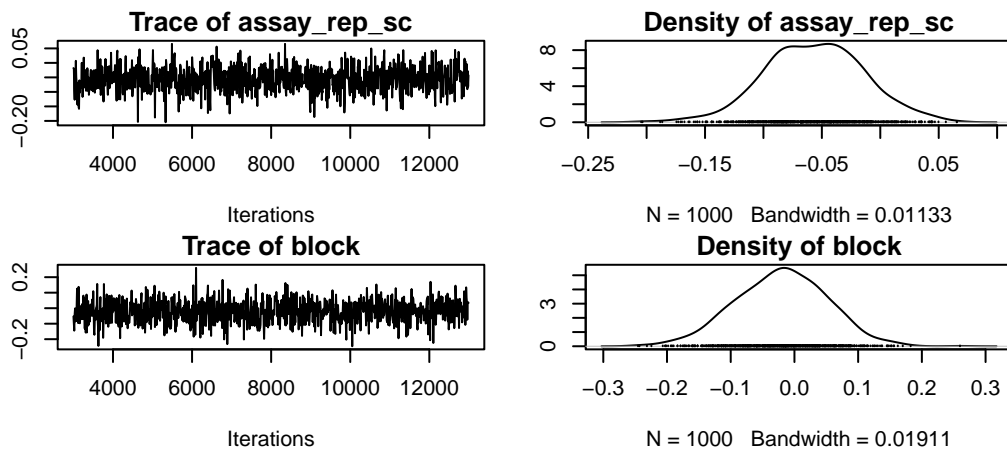


Figure 7.4.: Posterior trace and distribution of the parameters in `m_mcmcglmm` using default settings

```
plot(m_mcmcglmm$VCV)
```

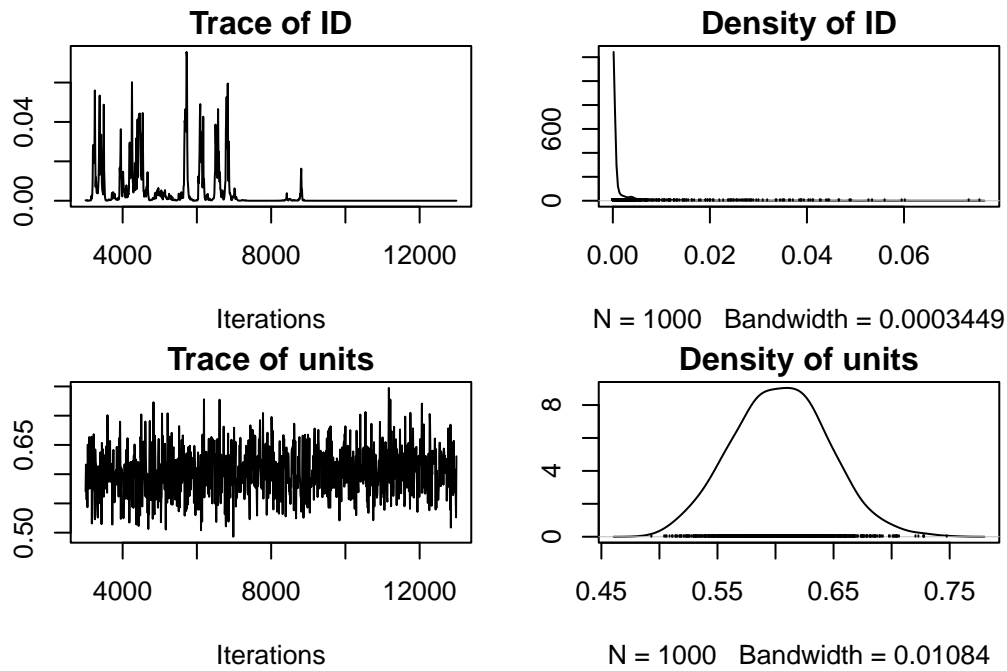



Figure 7.5.: Posterior trace and distribution of the parameters in `m_mcmcglmm` using default settings

```
par(omar)
autocorr.diag(m_mcmcglmm$VCV)
```

| | ID | units |
|---------|-----------|-------------|
| Lag 0 | 1.0000000 | 1.0000000 |
| Lag 10 | 0.8042405 | -0.02074155 |
| Lag 50 | 0.4807583 | -0.04264317 |
| Lag 100 | 0.1951356 | 0.04422296 |
| Lag 500 | 0.1254589 | 0.04401956 |

Talk about autocorrelation, mixing, convergence and priors here

```
n_samp <- 1000
thin <- 500
burnin <- 20000
mcglm_time <- system.time(
  m_mcmcglmm <- MCMCglmm(
    aggression ~ opp_size + body_size_sc + assay_rep_sc + block,
    random = ~ID,
    data = unicorns,
    nitt = n_samp * thin + burnin, thin = thin, burnin = burnin,
    verbose = FALSE,
    prior = list(
      R = list(V = 1, nu = 0.002),
```

```

      G = list(
        G1 = list(V = 1, nu = 0.002)
      )
    )
  )
summary(m_mcmcglmm)

```

```

Iterations = 20001:519501
Thinning interval = 500
Sample size = 1000

```

```
DIC: 1126.66
```

```
G-structure: ~ID
```

| | post.mean | l-95% CI | u-95% CI | eff.samp |
|----|-----------|-----------|----------|----------|
| ID | 0.01987 | 0.0002904 | 0.05458 | 1000 |

```
R-structure: ~units
```

| | post.mean | l-95% CI | u-95% CI | eff.samp |
|-------|-----------|----------|----------|----------|
| units | 0.5917 | 0.5188 | 0.6763 | 1000 |

```
Location effects: aggression ~ opp_size + body_size_sc + assay_rep_sc + block
```

| | post.mean | l-95% CI | u-95% CI | eff.samp | pMCMC |
|--------------|-----------|----------|----------|----------|------------|
| (Intercept) | 9.00136 | 8.92221 | 9.07383 | 1000 | <0.001 *** |
| opp_size | 1.05363 | 0.96382 | 1.13650 | 1000 | <0.001 *** |
| body_size_sc | 0.03373 | -0.03781 | 0.10686 | 1000 | 0.396 |
| assay_rep_sc | -0.05861 | -0.14186 | 0.02882 | 1000 | 0.182 |
| block | -0.02709 | -0.16061 | 0.11441 | 1000 | 0.698 |

```
---
```

```
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
mcglm_time
```

```

user  system elapsed
67.186  0.000  67.198

```

```
evaluate model here
```

```

omar <- par()
par(mar = c(4, 2, 1.5, 2))
plot(m_mcmcglmm$Sol)

```

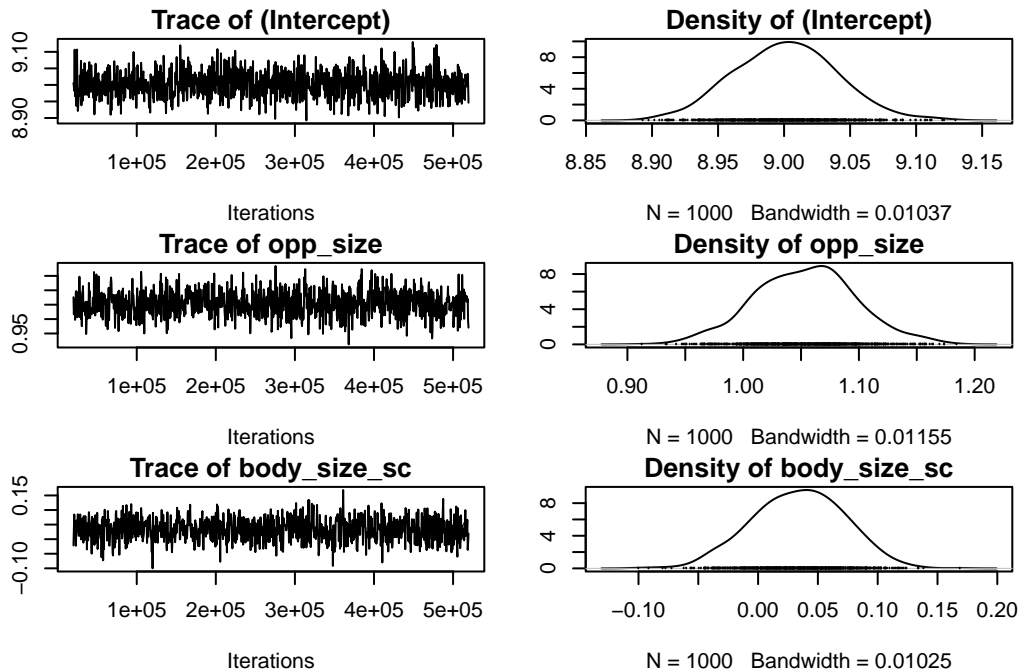


Figure 7.6.: Posterior trace and distribution of the parameters in `m_mcmcglmm` with better settings

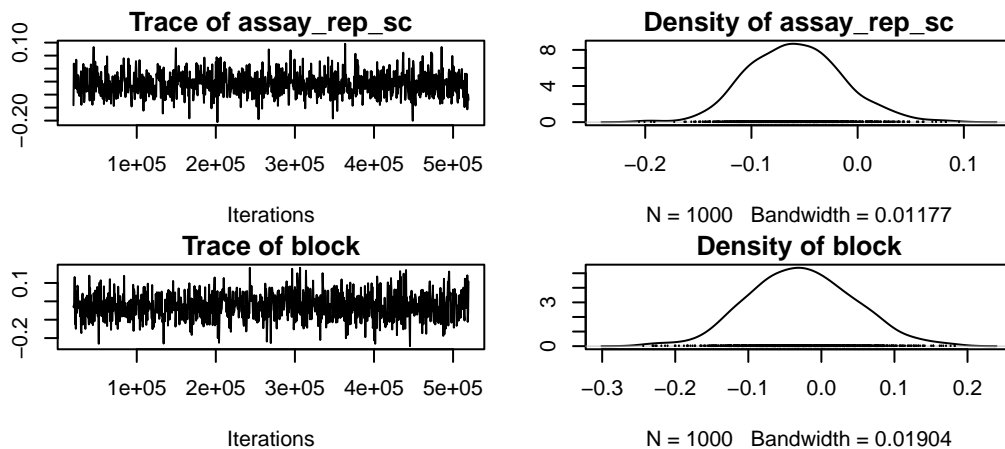


Figure 7.7.: Posterior trace and distribution of the parameters in `m_mcmcglmm` with better settings

```
plot(m_mcmcglmm$VCV)
```

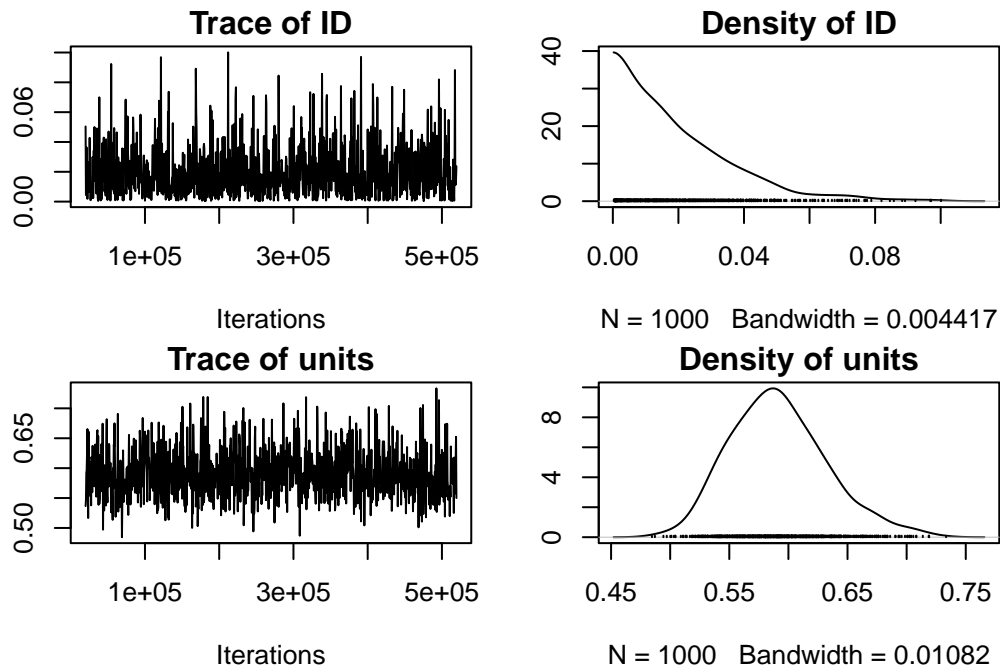


Figure 7.8.: Posterior trace and distribution of the parameters in `m_mcmcglmm` with better settings

```
par(omar)
autocorr.diag(m_mcmcglmm$VCV)
```

| | ID | units |
|-----------|--------------|--------------|
| Lag 0 | 1.000000000 | 1.000000000 |
| Lag 500 | 0.013876043 | -0.044235206 |
| Lag 2500 | 0.026120260 | -0.048012241 |
| Lag 5000 | -0.049357725 | 0.021158672 |
| Lag 25000 | 0.002544256 | -0.003722595 |

7.2.4. Inferences

7.2.4.1. Fixed effects

Easy peazy lemon squeezy just have a look at the posterior distribution, does it overlap 0 yes or no.

```
posterior.mode(m_mcmcglmm$Sol)
```

| (Intercept) | opp_size | body_size_sc | assay_rep_sc | block |
|-------------|------------|--------------|--------------|-------------|
| 9.00632282 | 1.07353252 | 0.03500916 | -0.04048582 | -0.03276275 |

```
HPDinterval(m_mcmcglmm$Sol)
```

```

              lower      upper
(Intercept)  8.92221005  9.07383400
opp_size     0.96382086  1.13649873
body_size_sc -0.03781276  0.10685606
assay_rep_sc -0.14185602  0.02882443
block        -0.16060691  0.11440706
attr(,"Probability")
[1] 0.95

```

7.2.4.2. Random effects

Quite a bit more harder. because constrained to be positive

```
posterior.mode(m_mcmcglmm$VCV)
```

```

      ID      units
0.00096263 0.59129362

```

```
HPDinterval(m_mcmcglmm$VCV)
```

```

              lower      upper
ID          0.0002903938  0.05458376
units      0.5188238599  0.67634529
attr(,"Probability")
[1] 0.95

```

7.2.5. brms

brms is an acronym for *Bayesian Regression Models using ‘Stan’* (Bürkner 2021). It is a package developed to fit regression models with a Bayesian approach using the amazing `stan` software (Stan Development Team 2021).

What makes `brms` so useful and powerful 🍷 in ecology is that:

1. it is really intuitive to code (same syntax as `glmer()`)
2. it is incredibly flexible since it is essentially a front end for `stan` via its `rstan` interface (Stan Development Team 2024)

but with *great powers come great responsibility* 🕷️

```
brm_time <- system.time(
  m_brm <- brm(
    aggression ~ opp_size + body_size_sc + assay_rep_sc + block
      + (1 | ID),
    data = unicorns, iter = 4750, warmup = 1000, thin = 15, cores = 4
    # refresh = 0
  )
)
```

Compiling Stan program...

Start sampling

```
brm_time
```

```
   user  system elapsed
103.424   6.753  93.875
```

```
summary(m_brm)
```

```
Family: gaussian
Links: mu = identity; sigma = identity
Formula: aggression ~ opp_size + body_size_sc + assay_rep_sc + block + (1 | ID)
Data: unicorns (Number of observations: 480)
Draws: 4 chains, each with iter = 4750; warmup = 1000; thin = 15;
       total post-warmup draws = 1000
```

Group-Level Effects:

~ID (Number of levels: 80)

| | Estimate | Est.Error | l-95% CI | u-95% CI | Rhat | Bulk_ESS | Tail_ESS |
|---------------|----------|-----------|----------|----------|------|----------|----------|
| sd(Intercept) | 0.14 | 0.07 | 0.01 | 0.27 | 1.00 | 1077 | 994 |

Population-Level Effects:

| | Estimate | Est.Error | l-95% CI | u-95% CI | Rhat | Bulk_ESS | Tail_ESS |
|--------------|----------|-----------|----------|----------|------|----------|----------|
| Intercept | 9.00 | 0.04 | 8.92 | 9.08 | 1.00 | 1028 | 918 |
| opp_size | 1.05 | 0.04 | 0.96 | 1.14 | 1.00 | 969 | 953 |
| body_size_sc | 0.03 | 0.04 | -0.05 | 0.12 | 1.00 | 1047 | 948 |
| assay_rep_sc | -0.06 | 0.04 | -0.15 | 0.03 | 1.00 | 870 | 994 |
| block | -0.02 | 0.07 | -0.16 | 0.11 | 1.00 | 778 | 945 |

Family Specific Parameters:

| | Estimate | Est.Error | l-95% CI | u-95% CI | Rhat | Bulk_ESS | Tail_ESS |
|--|----------|-----------|----------|----------|------|----------|----------|
|--|----------|-----------|----------|----------|------|----------|----------|

```
sigma      0.77      0.03      0.72      0.82 1.00      988      985
```

Draws were sampled using sampling(NUTS). For each parameter, Bulk_ESS and Tail_ESS are effective sample size measures, and Rhat is the potential scale reduction factor on split chains (at convergence, Rhat = 1).

```
mcmc_acf_bar(m_brm, regex_pars = c("sd"))
```

Warning: The `facets` argument of `facet_grid()` is deprecated as of ggplot2 2.2.0.
 i Please use the `rows` argument instead.
 i The deprecated feature was likely used in the bayesplot package.
 Please report the issue at <<https://github.com/stan-dev/bayesplot/issues/>>.

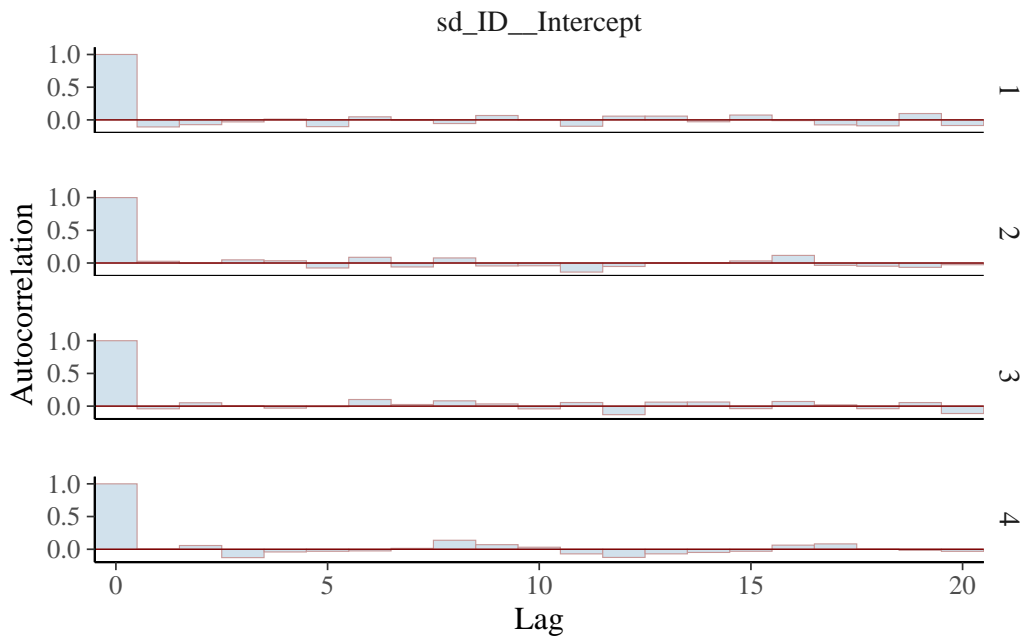


Figure 7.9.: Autocorrelation in the chain for variance parameters in model `m_brm`

7.2.5.1. Hunder the hood

have a look at the stan code

```
stancode(m_brm)
```

```
// generated with brms 2.20.4
functions {
}
```

```

data {
  int<lower=1> N; // total number of observations
  vector[N] Y; // response variable
  int<lower=1> K; // number of population-level effects
  matrix[N, K] X; // population-level design matrix
  int<lower=1> Kc; // number of population-level effects after centering
  // data for group-level effects of ID 1
  int<lower=1> N_1; // number of grouping levels
  int<lower=1> M_1; // number of coefficients per level
  array[N] int<lower=1> J_1; // grouping indicator per observation
  // group-level predictor values
  vector[N] Z_1_1;
  int prior_only; // should the likelihood be ignored?
}
transformed data {
  matrix[N, Kc] Xc; // centered version of X without an intercept
  vector[Kc] means_X; // column means of X before centering
  for (i in 2:K) {
    means_X[i - 1] = mean(X[, i]);
    Xc[, i - 1] = X[, i] - means_X[i - 1];
  }
}
parameters {
  vector[Kc] b; // regression coefficients
  real Intercept; // temporary intercept for centered predictors
  real<lower=0> sigma; // dispersion parameter
  vector<lower=0>[M_1] sd_1; // group-level standard deviations
  array[M_1] vector[N_1] z_1; // standardized group-level effects
}
transformed parameters {
  vector[N_1] r_1_1; // actual group-level effects
  real lprior = 0; // prior contributions to the log posterior
  r_1_1 = (sd_1[1] * (z_1[1]));
  lprior += student_t_lpdf(Intercept | 3, 8.9, 2.5);
  lprior += student_t_lpdf(sigma | 3, 0, 2.5)
    - 1 * student_t_lccdf(0 | 3, 0, 2.5);
  lprior += student_t_lpdf(sd_1 | 3, 0, 2.5)
    - 1 * student_t_lccdf(0 | 3, 0, 2.5);
}
model {
  // likelihood including constants
  if (!prior_only) {
    // initialize linear predictor term
    vector[N] mu = rep_vector(0.0, N);
    mu += Intercept;
    for (n in 1:N) {
      // add more terms to the linear predictor
      mu[n] += r_1_1[J_1[n]] * Z_1_1[n];
    }
    target += normal_id_glm_lpdf(Y | Xc, mu, b, sigma);
  }
}

```



```

}
// priors including constants
target += lprior;
target += std_normal_lpdf(z_1[1]);
}
generated quantities {
  // actual population-level intercept
  real b_Intercept = Intercept - dot_product(means_X, b);
}

```

7.2.5.2. using shiny

```
launch_shinystan(m_brm)
```

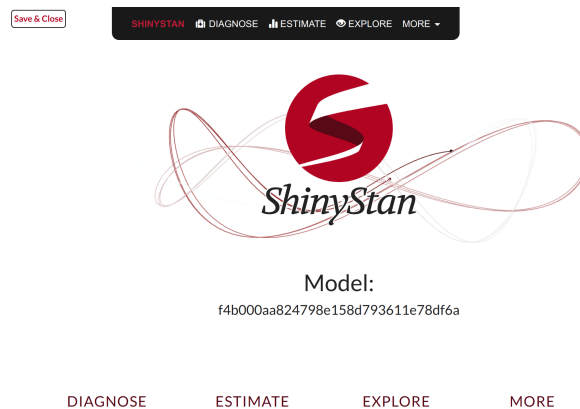


Figure 7.10.: ShinyStan interface

7.2.6. Inferences

7.2.6.1. Fixed effects

```
summary(m_brm)
```

```

Family: gaussian
Links: mu = identity; sigma = identity
Formula: aggression ~ opp_size + body_size_sc + assay_rep_sc + block + (1 | ID)
Data: unicorns (Number of observations: 480)
Draws: 4 chains, each with iter = 4750; warmup = 1000; thin = 15;

```

total post-warmup draws = 1000

Group-Level Effects:

~ID (Number of levels: 80)

| | Estimate | Est.Error | l-95% CI | u-95% CI | Rhat | Bulk_ESS | Tail_ESS |
|---------------|----------|-----------|----------|----------|------|----------|----------|
| sd(Intercept) | 0.14 | 0.07 | 0.01 | 0.27 | 1.00 | 1077 | 994 |

Population-Level Effects:

| | Estimate | Est.Error | l-95% CI | u-95% CI | Rhat | Bulk_ESS | Tail_ESS |
|--------------|----------|-----------|----------|----------|------|----------|----------|
| Intercept | 9.00 | 0.04 | 8.92 | 9.08 | 1.00 | 1028 | 918 |
| opp_size | 1.05 | 0.04 | 0.96 | 1.14 | 1.00 | 969 | 953 |
| body_size_sc | 0.03 | 0.04 | -0.05 | 0.12 | 1.00 | 1047 | 948 |
| assay_rep_sc | -0.06 | 0.04 | -0.15 | 0.03 | 1.00 | 870 | 994 |
| block | -0.02 | 0.07 | -0.16 | 0.11 | 1.00 | 778 | 945 |

Family Specific Parameters:

| | Estimate | Est.Error | l-95% CI | u-95% CI | Rhat | Bulk_ESS | Tail_ESS |
|-------|----------|-----------|----------|----------|------|----------|----------|
| sigma | 0.77 | 0.03 | 0.72 | 0.82 | 1.00 | 988 | 985 |

Draws were sampled using sampling(NUTS). For each parameter, Bulk_ESS and Tail_ESS are effective sample size measures, and Rhat is the potential scale reduction factor on split chains (at convergence, Rhat = 1).

```
mcmc_plot(m_brm, regex_pars = "b_")
```

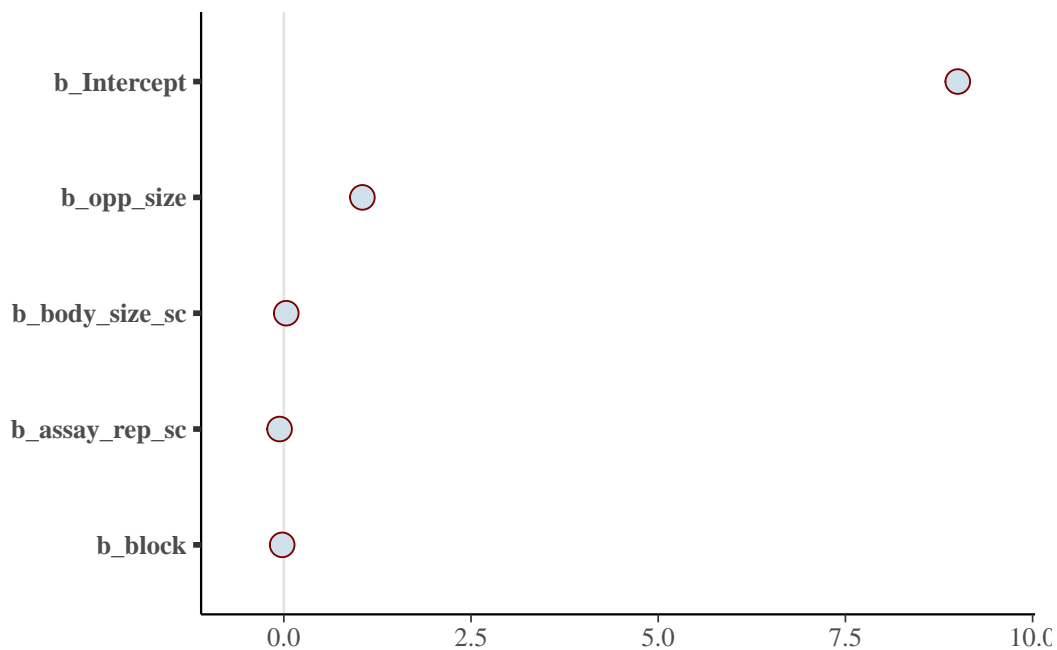


Figure 7.11.: Fixed effect estimates (with 95% credible intervals) from model m_brm

7.2.6.2. Random effects

```
summary(m_brm)
```

```
Family: gaussian
Links: mu = identity; sigma = identity
Formula: aggression ~ opp_size + body_size_sc + assay_rep_sc + block + (1 | ID)
Data: unicorns (Number of observations: 480)
Draws: 4 chains, each with iter = 4750; warmup = 1000; thin = 15;
       total post-warmup draws = 1000
```

Group-Level Effects:

```
~ID (Number of levels: 80)
```

| | Estimate | Est.Error | l-95% CI | u-95% CI | Rhat | Bulk_ESS | Tail_ESS |
|---------------|----------|-----------|----------|----------|------|----------|----------|
| sd(Intercept) | 0.14 | 0.07 | 0.01 | 0.27 | 1.00 | 1077 | 994 |

Population-Level Effects:

| | Estimate | Est.Error | l-95% CI | u-95% CI | Rhat | Bulk_ESS | Tail_ESS |
|--------------|----------|-----------|----------|----------|------|----------|----------|
| Intercept | 9.00 | 0.04 | 8.92 | 9.08 | 1.00 | 1028 | 918 |
| opp_size | 1.05 | 0.04 | 0.96 | 1.14 | 1.00 | 969 | 953 |
| body_size_sc | 0.03 | 0.04 | -0.05 | 0.12 | 1.00 | 1047 | 948 |
| assay_rep_sc | -0.06 | 0.04 | -0.15 | 0.03 | 1.00 | 870 | 994 |
| block | -0.02 | 0.07 | -0.16 | 0.11 | 1.00 | 778 | 945 |

Family Specific Parameters:

| | Estimate | Est.Error | l-95% CI | u-95% CI | Rhat | Bulk_ESS | Tail_ESS |
|-------|----------|-----------|----------|----------|------|----------|----------|
| sigma | 0.77 | 0.03 | 0.72 | 0.82 | 1.00 | 988 | 985 |

Draws were sampled using sampling(NUTS). For each parameter, Bulk_ESS and Tail_ESS are effective sample size measures, and Rhat is the potential scale reduction factor on split chains (at convergence, Rhat = 1).

```
mcmc_plot(m_brm, pars = c("sd_ID__Intercept", "sigma"))
```

Warning: Argument 'pars' is deprecated. Please use 'variable' instead.

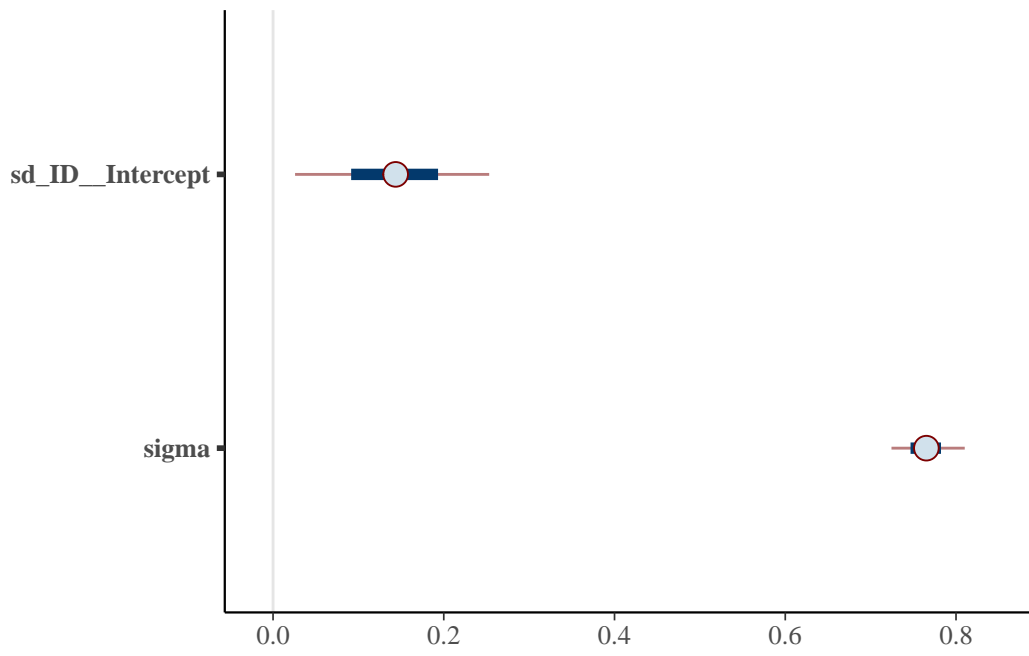


Figure 7.12.: Among-individual and residual standard deviance (with 95% credible intervals) estimated from model `m_brm`

7.2.7. Happy Bayesian stats



Figure 7.13.: Sherlock Holmes, a truly bayesian detective

8. Multivariate mixed models

8.1. Lecture

Amazing beasts and crazy animals





Figure 8.1.: Dream pet dragon

add a comparison of lrt

8.2. Practical

In this practical, we have collected data on the amazing blue dragon of the East that roam the sky at night.

We will use two different  to fit more complex models that are not possible with `lmer()` from `lme4`  (Bates et al. 2015). We will use:

- `asreml-R` which is a commercial software developed by VSNi (The VSNi Team 2023). `ASReml` fit models using a maximum likelihood approach, is quite flexible and fast.
- `MCMCglmm` which is free and open-source and fit model using a Bayesian approach (Hadfield 2010). It is super flexible and allow to fit a wide diversity of distribution.

The aims of the practical are to learn:

- How to phrase questions of interest in terms of variances and covariances (or derived correlations or regressions);
- How to incorporate more advanced model structures, such as:
 - Fixed effects that apply only to a subset of the response traits;

- Traits which are measured a different number of times (e.g., repeated measures of behaviour and a single value of breeding success);
- Hypothesis testing using likelihood ratio tests.

8.2.1. R packages needed

First we load required libraries

```
library(lmerTest)
library(tidyverse)
library(asreml)
library(MCMCglmm)
library(nadiv)
```

8.2.2. The blue dragon of the East

For this practical, we have collected data on the amazing blue dragon of the East that roam the sky at night.



Figure 8.2.: Blue dragon male

We tagged all dragons individually when they hatch from their eggs. Here, we concentrate on female dragon that produce a single clutch of eggs per mating seasons. Adult females blue dragons need to explore vast amount of land to find a compatible male. We thus hypothesized that maximum flight speed as well as exploration are key traits to determine fitness. We were able to obtain repeated measures of flying speed and exploration on 80 adult females during one mating season and also measure the number of egg laid at the end of the season.

Each females was capture 4 times during the season and each time we measured the maximum flying speed using a wind tunnel and exploration using a openfield test.

The data frame has 6 variables:

- ID: Individual identity
- assay_rep: the repeat number of the behavioural assay
- max_speed: maximum flying speed
- exploration:

- `eggs`: measure of reproductive success measured only once per individual
- `body_size`: individual body size measured on the day of the test


```
df_dragons <- read.csv("data/dragons.csv")
str(df_dragons)
```

```
'data.frame':  320 obs. of  6 variables:
 $ ID      : chr  "S_1" "S_1" "S_1" "S_1" ...
 $ assay_rep : int  1 2 3 4 1 2 3 4 1 2 ...
 $ max_speed : num  58.7 57.9 64.3 61.4 65.5 ...
 $ exploration: num  126 125 127 127 125 ...
 $ eggs     : int  39 NA NA NA 56 NA NA NA 51 NA ...
 $ body_size : num  21.7 21.5 21.3 20.8 25.7 ...
```

To help with convergence of the model, and also help with parameter interpretation, we will first scale our covariates.

```
df_dragons <- df_dragons %>%
  mutate(
    body_size_sc = scale(body_size),
    assay_rep_sc = scale(assay_rep, scale = FALSE)
  )
```

8.2.3. Multiple univariate models

We first use the `lme4`  to determine the proportion of phenotypic variation (adjusted for fixed effects) that is due to differences among individuals, separately for each trait with repeated measures.

8.2.3.1. Flying speed

Our model includes fixed effects of the assay repeat number (centred) and individual body size (centred and scaled to standard deviation units), as we wish to control for any systematic effects of these variables on individual behaviour. Be aware that controlling variables are at your discretion — for example, while we want to characterise among-individual variance in flying speed after controlling for size effects in this study, others may wish to characterise among-individual variance in flying speed without such control. Using techniques shown later in the practical, it would be entirely possible to characterise both among-individual variance in flying speed and in size, and the among-individual covariance between these measurements.

```
lmer_f <- lmer(max_speed ~ assay_rep_sc + body_size_sc + (1 | ID),
  data = df_dragons
)
par(mfrow = c(1, 3))
plot(resid(lmer_f, type = "pearson") ~ fitted(lmer_f))
qqnorm(residuals(lmer_f))
qqline(residuals(lmer_f))
hist(residuals(lmer_f))
```

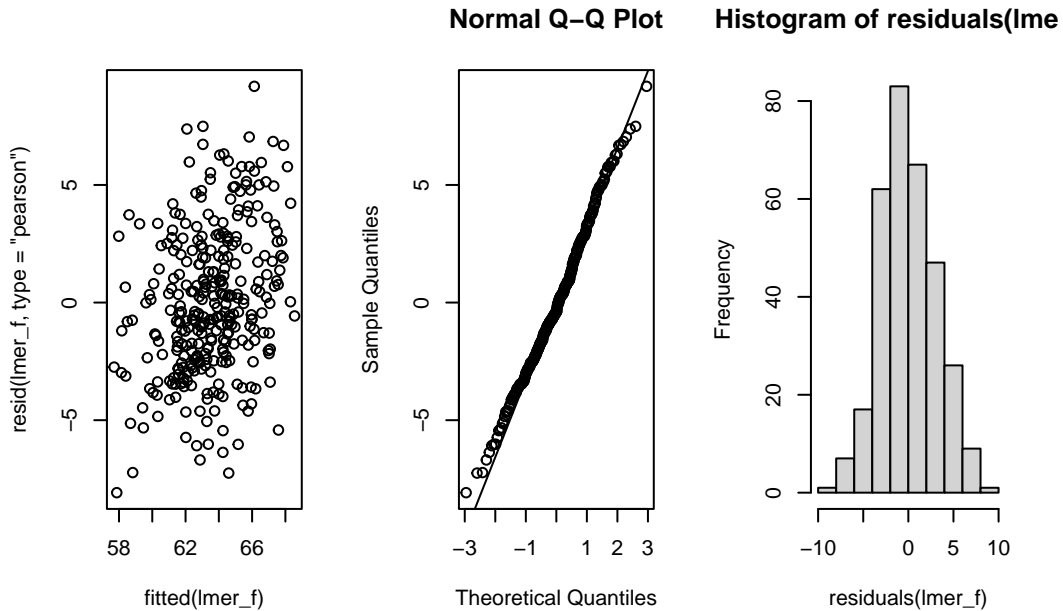


Figure 8.3.: Checking assumptions of model lmer_f

```
summary(lmer_f)
```

```
Linear mixed model fit by REML. t-tests use Satterthwaite's method [
lmerModLmerTest]
```

```
Formula: max_speed ~ assay_rep_sc + body_size_sc + (1 | ID)
```

```
Data: df_dragons
```

```
REML criterion at convergence: 1791.4
```

```
Scaled residuals:
```

```
      Min       1Q   Median       3Q      Max
-2.3645 -0.6496 -0.1154  0.6463  2.6894
```

```
Random effects:
```

```
Groups   Name             Variance Std.Dev.
ID       (Intercept)    6.951   2.636
Residual                    11.682   3.418
```

```
Number of obs: 320, groups: ID, 80
```

```
Fixed effects:
```

```
              Estimate Std. Error      df t value Pr(>|t|)
(Intercept)  63.5344    0.3513  78.0954 180.870 <2e-16 ***
assay_rep_sc -0.1519    0.1709 238.9807  -0.889   0.375
body_size_sc  0.4468    0.3445  88.0328   1.297   0.198
```

```
---
```

```
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Correlation of Fixed Effects:

```
(Intr) assy__
assay_rp_sc 0.000
body_siz_sc 0.000 -0.002
```

Having examined diagnostic plots of the model fit, we can check the model summary. We are interested in the random effects section of the lme4 model output (specifically the variance component — note that the standard deviation here is simply the square root of the variance). Evidence for ‘animal personality’ (or ‘consistent among-individual differences in behaviour’) in the literature is largely taken from the repeatability of behavioural traits: we can compute this repeatability (also known as the intraclass correlation coefficient) by dividing the variance in the trait due to differences among individuals (V_{ID}) by the total phenotypic variance after accounting for the fixed effects ($V_{ID} + V_{residual}$).

```
rep_flying <- as.data.frame(VarCorr(lmer_f)) %>%
  select(grp, vcov) %>%
  spread(grp, vcov) %>%
  mutate(repeatability = ID / (ID + Residual))
rep_flying
```

Table 8.1.: Variance components and repeatability for the maximum flying speed of blue dragons

| ID | Residual | repeatability |
|-------|----------|---------------|
| 6.951 | 11.682 | 0.373 |

So we can see that 37.31% of the phenotypic variation in boldness (having controlled for body size and assay repeat number) is due to differences among individuals.

8.2.3.2. Exploration

```
lmer_e <- lmer(exploration ~ assay_rep_sc + body_size_sc + (1 | ID),
  data = df_dragons
)
par(mfrow = c(1, 3))
plot(resid(lmer_e, type = "pearson") ~ fitted(lmer_e))
qqnorm(residuals(lmer_e))
qqline(residuals(lmer_e))
hist(residuals(lmer_e))
```

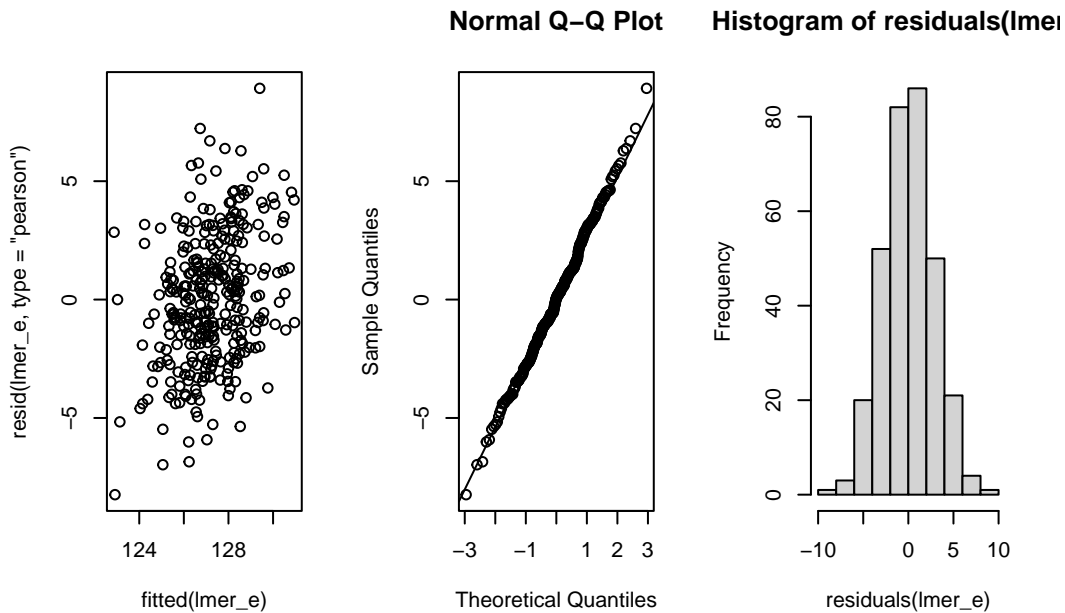


Figure 8.4.: Checking assumptions of model `lmer_e`

```
summary(lmer_e)
```

```
Linear mixed model fit by REML. t-tests use Satterthwaite's method [
lmerModLmerTest]
```

```
Formula: exploration ~ assay_rep_sc + body_size_sc + (1 | ID)
```

```
Data: df_dragons
```

```
REML criterion at convergence: 1691.2
```

```
Scaled residuals:
```

| Min | 1Q | Median | 3Q | Max |
|----------|----------|---------|---------|---------|
| -2.73290 | -0.62520 | 0.01635 | 0.55523 | 2.95896 |

```
Random effects:
```

| Groups | Name | Variance | Std.Dev. |
|--------|-------------|----------|----------|
| ID | (Intercept) | 3.623 | 1.903 |
| | Residual | 9.091 | 3.015 |

```
Number of obs: 320, groups: ID, 80
```

```
Fixed effects:
```

| | Estimate | Std. Error | df | t value | Pr(> t) |
|--------------|-----------|------------|-----------|---------|------------|
| (Intercept) | 127.22524 | 0.27148 | 78.08871 | 468.639 | <2e-16 *** |
| assay_rep_sc | -0.07811 | 0.15076 | 238.99943 | -0.518 | 0.605 |
| body_size_sc | 0.26114 | 0.26806 | 85.68180 | 0.974 | 0.333 |

```
---
```

```
Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Correlation of Fixed Effects:

```
(Intr) assy__
assay_rp_sc 0.000
body_siz_sc 0.000 -0.002
```

So the model looks good and we can see our estimates for both fixed and random effects. We can now estimate the repeatability of exploration.

```
rep_expl <- as.data.frame(VarCorr(lmer_e)) %>%
  select(grp, vcov) %>%
  spread(grp, vcov) %>%
  mutate(repeatability = ID / (ID + Residual))
rep_expl
```

Table 8.2.: Variance components and repeatability for exploration behaviour of blue dragons

| ID | Residual | repeatability |
|-------|----------|---------------|
| 3.623 | 9.091 | 0.285 |

Both of traits of interest are repeatable at the among-individual level. So, the remaining question is estimating the relation between these two traits. Are individuals that are consistently faster than average also more exploratory than average (and vice versa)?

8.2.3.3. Correlation using BLUPs

Using BLUPs to estimate correlations between traits or to further investigate biological associations can lead to spurious results and anticonservative hypothesis tests and narrow confidence intervals. Hadfield et al. (2010) discuss the problem as well as present some alternative method to avoid the problem using Bayesian methods. However, it is always preferable to use multivariate models when possible.

We need to create a data frame that contain the BLUPs from both univariate models.

```
df_blups_fe <- merge(
  as.data.frame(ranef(lmer_f)),
  as.data.frame(ranef(lmer_e)),
  by = "grp"
) %>%
  mutate(
    speed = condval.x,
    exploration = condval.y
  )
```

We can now test the correlation among-individual between flying speed and exploration.

```
(cor_blups <- with(df_blups_fe, cor.test(speed, exploration)))
```

Pearson's product-moment correlation

```
data: speed and exploration
t = 3.2131, df = 78, p-value = 0.00191
alternative hypothesis: true correlation is not equal to 0
95 percent confidence interval:
 0.1320924 0.5223645
sample estimates:
      cor
0.3418867
```

```
ggplot(df_blups_fe, aes(x = exploration, y = speed)) +
  geom_point() +
  labs(xlab = "Exploration (BLUP)", ylab = "Flying speed (BLUP)") +
  theme_classic()
```

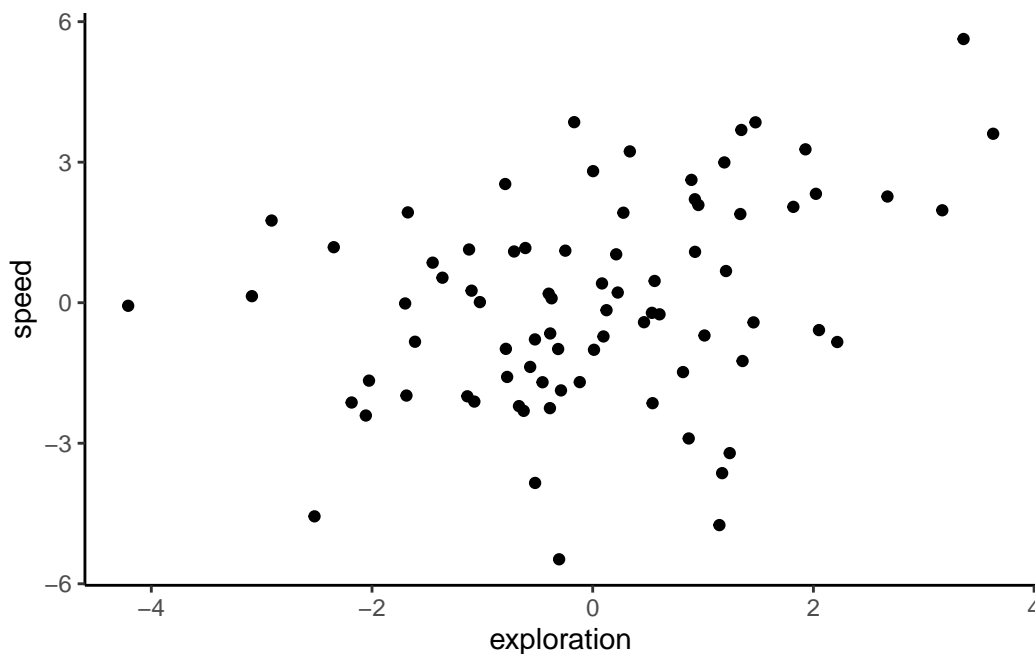


Figure 8.5.: Relation between exploration and flying speed using BLUPs from univariate models

As you can see, we get a positive correlation with a very small p-value ($P = 0.00191$), indicating that these traits are involved in a behavioural syndrome. While the correlation itself is fairly weak ($r = 0.342$), it appears to be highly significant, and suggests that individuals that are faster than average also tend to be more exploratory than average. However, as discussed in Hadfield et al. (2010) and Housley and Wilson (2017), using BLUPs in this way leads to

anticonservative significance tests. This is because the error inherent in their prediction is not carried forward from the lmer models to the subsequent analysis (in this case, a correlation test). To illustrate this point quickly, below we plot the individual estimates along with their associated standard errors.

```
ggplot(df_blups_fe, aes(x = exploration, y = speed)) +
  geom_point() +
  geom_linerange(aes(
    xmin = exploration - condsd.x,
    xmax = exploration + condsd.x
  )) +
  geom_linerange(aes(
    ymin = speed - condsd.y,
    ymax = speed + condsd.y
  )) +
  labs(
    xlab = "Exploration (BLUP +/- SE)",
    ylab = "Flying speed (BLUP +/- SE)"
  ) +
  theme_classic()
```

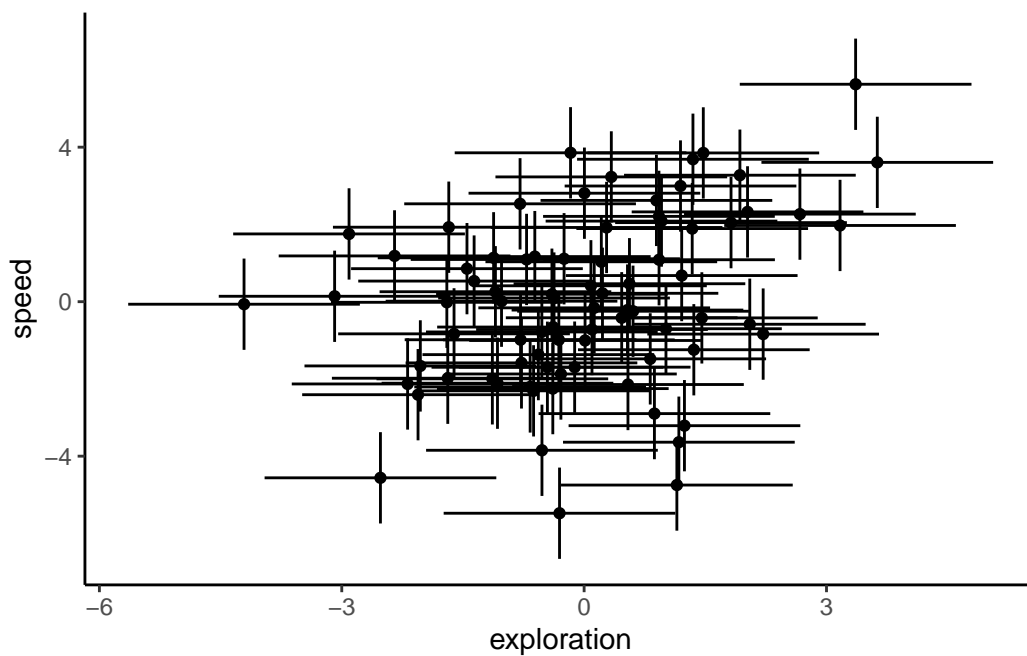


Figure 8.6.: Relation between exploration and flying speed using BLUPs from univariate models including +/- SE as error bars

8.2.4. Multivariate approach

8.2.4.1. Based on ASRemlR

The correct approach for testing the hypothesised relation between speed and exploration uses both response variables in a two-trait (‘bivariate’) mixed model. This model estimates the among-individual variance for each response variable (and the covariance between them). Separate (co)variances are also fitted for the residual variation. The bivariate model also allows for fixed effects to be fitted on both response variables. We set up our model using the `asreml` function call, with our bivariate response variable being `exploration` and `flying speed` bound together using `cbind`. You will also note that we scale our response variables, meaning that each is centred at their mean value and standardised to units of 1 standard deviation. This is not essential, but simply makes it easier for the model to be fit. Scaling the response variables also aids our understanding of the output, as both `flying speed` and `exploration` are now on the same scale.

`asreml` can be a bit specific sometime and random effects should absolutely be `factor` and not `character` or `integer`

```
df_dragons <- df_dragons %>%
  mutate(
    ID = as.factor(ID),
    speed_sc = scale(max_speed),
    exploration_sc = scale(exploration)
  )

asr_us <- asreml(
  cbind(speed_sc, exploration_sc) ~ trait +
    trait:assay_rep_sc + trait:body_size_sc,
  random = ~ ID:us(trait),
  residual = ~ units:us(trait),
  data = df_dragons,
  maxiter = 100
)
```

Model fitted using the sigma parameterization.

```
ASReml 4.1.0 Thu Feb 1 15:38:07 2024
```

| | LogLik | Sigma2 | DF | wall | cpu |
|---|----------|--------|-----|----------|-----|
| 1 | -333.105 | 1.0 | 634 | 15:38:07 | 0.0 |
| 2 | -303.637 | 1.0 | 634 | 15:38:07 | 0.0 |
| 3 | -274.849 | 1.0 | 634 | 15:38:07 | 0.0 |
| 4 | -260.243 | 1.0 | 634 | 15:38:07 | 0.0 |
| 5 | -256.118 | 1.0 | 634 | 15:38:07 | 0.0 |
| 6 | -255.891 | 1.0 | 634 | 15:38:07 | 0.0 |
| 7 | -255.889 | 1.0 | 634 | 15:38:07 | 0.0 |

On the right hand side of our model formula, we use the `trait` keyword to specify that this is a multivariate model — `trait` itself tells the model to give us the intercept for each trait. We then interact `trait` with the fixed effects, `assay_rep_sc` and `body_size_sc`, so that we get estimates for the effect of these variables on each of the 2 traits. The random effects structure starts with the random effects, where we tell the model to fit an *unstructured* (`us`)

covariance matrix for the grouping variable ID. This means that the variance in exploration due to differences among individuals, the variance in boldness due to differences among individuals, and the covariance between these variances will be estimated. Next, we set a structure for the residual variation (`residual`), which is also sometimes known as the *within-individual variation*. As we have repeated measures for both traits at the individual level, we also set an *unstructured* covariance matrix, which estimates the residual variance for each trait and also allows the residuals to covary across the two traits. Finally, we provide the name of the data frame, and a maximum number of iterations for ASReml to attempt to fit the model. After the model has been fit by ASReml, we can check the fit using the same type of model diagnostic plots as we use for `lme4`:

```
par(mfrow = c(1, 3))
plot(residuals(asr_us) ~ fitted(asr_us))
qqnorm(residuals(asr_us))
qqline(residuals(asr_us))
hist(residuals(asr_us))
```

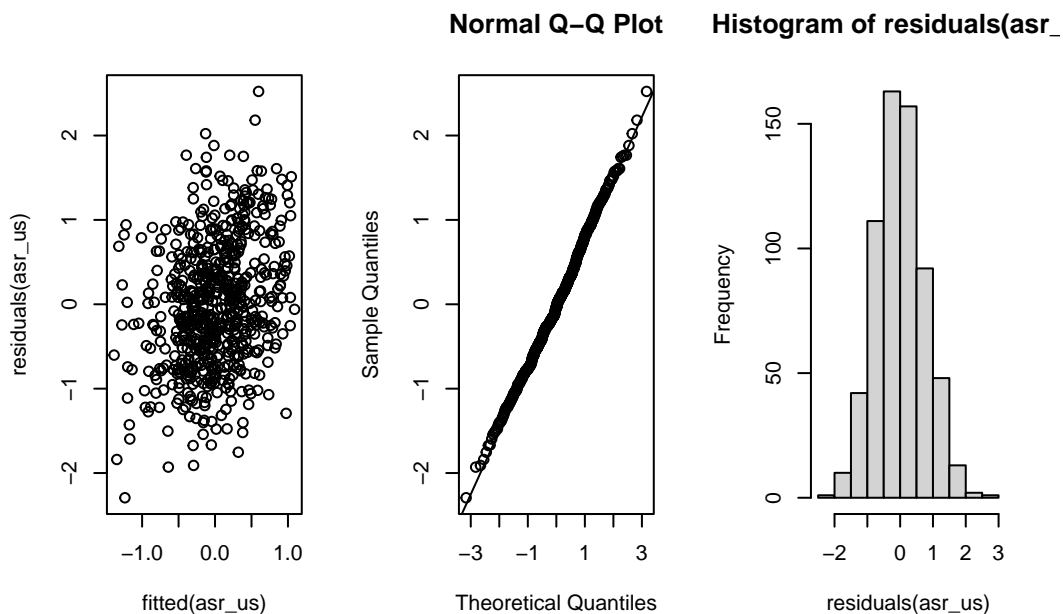


Figure 8.7.: Checking assumptions of model `asr_us`

The summary part of the ASReml model fit contains a large amount of information, so it is best to look only at certain parts of it at a single time. While we are not particularly interested in the fixed effects for current purposes, you can inspect these using the following code to check whether there were any large effects of `assay` `repeat` or `body_size` on either trait:

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

| | solution | std error | z.ratio |
|-----------------------------|--------------|------------|--------------|
| trait_speed_sc:body_size_sc | 1.040579e-01 | 0.07972962 | 1.305135e+00 |

```

trait_exploration_sc:body_size_sc  7.269022e-02  0.07533421  9.649033e-01
trait_speed_sc:assay_rep_sc        -3.521261e-02  0.03960492 -8.890967e-01
trait_exploration_sc:assay_rep_sc -2.195541e-02  0.04238056 -5.180538e-01
trait_speed_sc                     -1.820461e-16  0.08140684 -2.236251e-15
trait_exploration_sc               -2.853753e-16  0.07631479 -3.739449e-15

```

```
wa <- wald(asr_us, ssType = "conditional", denDF = "numeric")
```

Model fitted using the sigma parameterization.

ASReml 4.1.0 Thu Feb 1 15:38:08 2024

| | LogLik | Sigma2 | DF | wall | cpu |
|---|----------|--------|-----|----------|-----|
| 1 | -255.889 | 1.0 | 634 | 15:38:08 | 0.0 |
| 2 | -255.889 | 1.0 | 634 | 15:38:08 | 0.0 |

Calculating denominator DF

```
attr(wa$Wald, "heading") <- NULL
wa
```

\$Wald

| | Df | denDF | F.inc | F.con | Margin | Pr |
|--------------------|----|-------|--------|--------|--------|---------|
| trait | 2 | 77.1 | 0.0000 | 0.0000 | | 1.00000 |
| trait:assay_rep_sc | 2 | 237.9 | 0.3955 | 0.3984 | B | 0.67184 |
| trait:body_size_sc | 2 | 86.6 | 0.9871 | 0.9871 | B | 0.37679 |

\$stratumVariances

NULL

We can see that there is a separate intercept for both personality traits (no surprise that these are very close to zero, given that we mean-centred and scaled each trait before fitting the model), and an estimate of the effect of assay repeat and body size on both traits. None of these appear to be large effects, so let's move on to the more interesting parts — the random effects estimates:

```
summary(asr_us)$varcomp
```

| | component | std.error | z.ratio |
|--|------------|------------|-----------|
| ID:trait!trait_speed_sc:speed_sc | 0.37333063 | 0.08607123 | 4.337461 |
| ID:trait!trait_exploration_sc:speed_sc | 0.08838639 | 0.06067006 | 1.456837 |
| ID:trait!trait_exploration_sc:exploration_sc | 0.28631012 | 0.07637247 | 3.748865 |
| units:trait!R | 1.00000000 | NA | NA |
| units:trait!trait_speed_sc:speed_sc | 0.62741689 | 0.05740281 | 10.930073 |
| units:trait!trait_exploration_sc:speed_sc | 0.32632113 | 0.04829175 | 6.757286 |

```

units:trait!trait_exploration_sc:exploration_sc 0.71844189 0.06572780 10.930563
                                         bound %ch
ID:trait!trait_speed_sc:speed_sc          P 0
ID:trait!trait_exploration_sc:speed_sc     P 0
ID:trait!trait_exploration_sc:exploration_sc P 0
units:trait!R                              F 0
units:trait!trait_speed_sc:speed_sc       P 0
units:trait!trait_exploration_sc:speed_sc  P 0
units:trait!trait_exploration_sc:exploration_sc P 0

```

In the above summary table, we have the among-individual (co)variances listed first (starting with ID), then the residual (or within-individual) (co)variances (starting with R). You will notice that the variance estimates here are actually close to the lme4 repeatability estimates, because our response variables were scaled to phenotypic standard deviations. We can also find the ‘adjusted repeatability’ (i.e., the repeatability conditional on the fixed effects) for each trait by dividing its among-individual variance estimate by the sum of its among-individual and residual variances. Here, we use the `vpredict` function to estimate the repeatability and its standard error for each trait, conditional on the effects of assay repeat and body size. For this function, we provide the name of the model object, followed by a name that we want to give the estimate being returned, and a formula for the calculation. Each ‘V’ term in the formula refers to a variance component, using its position in the model summary shown above.

```
vpredict(asr_us, rep_speed ~ V1 / (V1 + V5))
```

```

          Estimate      SE
rep_speed 0.3730518 0.06124032

```

```
vpredict(asr_us, rep_expl ~ V3 / (V3 + V7))
```

```

          Estimate      SE
rep_expl 0.284956 0.06113539

```

We can also use this function to calculate the estimate and standard error of the correlation from our model (co)variances. We do this by specifying the formula for the correlation:

```
(cor_fe <- vpredict(asr_us, cor_expl_speed ~ V2 / (sqrt(V1 * V3))))
```

```

          Estimate      SE
cor_expl_speed 0.2703462 0.1594097

```

In this case, the estimate is similar (here, slightly lower) than our correlation estimate using BLUPs. However, if we consider confidence intervals as $\pm 1.96 SE$ around the estimate, the lower bound of the confidence interval would actually be -0.0421. With confidence intervals straddling zero, we would conclude that this correlation is likely non-significant. As the use of standard errors in this way is only approximate, we should also test our hypothesis formally using likelihood ratio tests.

8.2.4.1.1. Hypothesis testing

We can now test the statistical significance of this correlation directly, by fitting a second model without the among-individual covariance between our two traits, and then using a likelihood ratio test to determine whether the model with the covariance produces a better fit. Here, we use the `idh` structure for our random effects. This stands for ‘identity matrix’ (i.e., with 0s on the off-diagonals) with heterogeneous variances (i.e., the variance components for our two response traits are allowed to be different from one another). The rest of the model is identical to the previous version.

```
asr_idh <- asreml(
  cbind(speed_sc, exploration_sc) ~ trait +
    trait:assay_rep_sc + trait:body_size_sc,
  random = ~ ID:idh(trait),
  residual = ~ units:us(trait),
  data = df_dragons,
  maxiter = 100
)
```

Model fitted using the sigma parameterization.

ASReml 4.1.0 Thu Feb 1 15:38:08 2024

| | LogLik | Sigma2 | DF | wall | cpu |
|---|----------|--------|-----|----------|-----|
| 1 | -327.051 | 1.0 | 634 | 15:38:08 | 0.0 |
| 2 | -299.874 | 1.0 | 634 | 15:38:08 | 0.0 |
| 3 | -273.689 | 1.0 | 634 | 15:38:08 | 0.0 |
| 4 | -260.838 | 1.0 | 634 | 15:38:08 | 0.0 |
| 5 | -257.331 | 1.0 | 634 | 15:38:08 | 0.0 |
| 6 | -257.120 | 1.0 | 634 | 15:38:08 | 0.0 |
| 7 | -257.118 | 1.0 | 634 | 15:38:08 | 0.0 |

The likelihood ratio test is calculated as twice the difference between model log-likelihoods, on a single degree of freedom (the covariance term):

```
(p_biv <- pchisq(2 * (asr_us$loglik - asr_idh$loglik),
  df = 1,
  lower.tail = FALSE
))
```

```
[1] 0.1170385
```

In sharp contrast to the highly-significant P-value given by a correlation test using BLUPs, here we find no evidence for a correlation between flying speed and exploration. To better understand why BLUPs produce an anticonservative p-value in comparison to multivariate models, we should plot the correlation estimates and their confidence intervals. The confidence intervals are taken directly from the `cor.test` function for BLUPs, and for ASReml they are calculated as 1.96 times the standard error from the `vpredict` function.

```
df_cor <- data.frame(
  Method = c("ASReml", "BLUPs"),
  Correlation = c(as.numeric(cor_fe[1]), cor_blups$estimate),
  low = c(as.numeric(cor_fe[1] - 1.96 * cor_fe[2]), cor_blups$conf.int[1]),
  high = c(as.numeric(cor_fe[1] + 1.96 * cor_fe[2]), cor_blups$conf.int[2])
)
ggplot(df_cor, aes(x = Method, y = Correlation)) +
  geom_point() +
  geom_linerange(aes(ymin = low, ymax = high)) +
  ylim(-1, 1) +
  geom_hline(yintercept = 0, linetype = 2) +
  theme_classic()
```

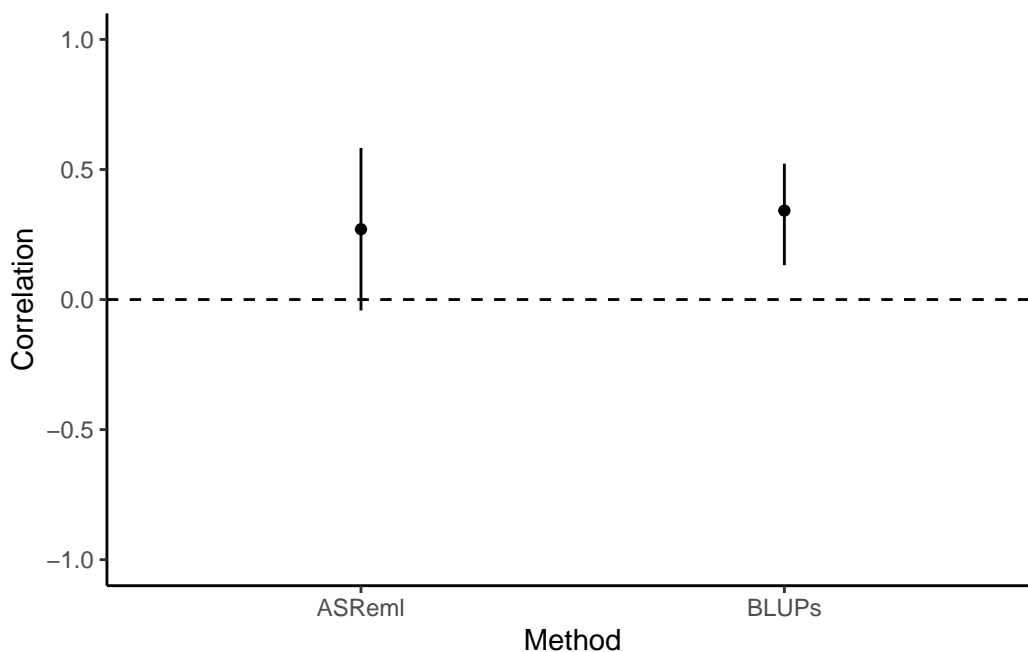


Figure 8.8.: Correlation estimates (with CI) using 2 different methods

Here we can clearly see that the BLUPs method - having failed to carry through the error around the predictions of individual-level estimates - is anticonservative, with small confidence intervals and a correspondingly small P-value ($P = 0.00191$). Testing the syndrome directly in a bivariate model that retains all the data, by comparison, enables us to capture the true uncertainty about the estimate of the correlation. This is reflected in the larger confidence intervals and, in this case, the non-significant P-value ($P = 0.117$).

8.2.4.1.2. Conclusions

To conclude, then: we found that the correlation between flying speed and exploration tends to be positive among female blue dragon. This correlation is not statistically significant, and thus does not provide strong evidence. However, inappropriate analysis of BLUP extracted from univariate models would lead to a different (erroneous) conclusion.

8.2.4.2. Using MCMCglmm

In this section I present the code needed to fit the model and explain only the specific aspect of fitting and evaluating the models with MCMCglmm.

To be completed. with more details

First, we need to create a ‘prior’ for our model. We recommend reading up on the use of priors (see the course notes of MCMCglmm Hadfield 2010); briefly, we use a parameter-expanded prior here that should be uninformative for our model. One of the model diagnostic steps that should be used later is to check that the model is robust to multiple prior specifications.

```
prior_1ex <- list(
  R = list(V = diag(2), nu = 0.002),
  G = list(G1 = list(
    V = diag(2) * 0.02, nu = 3,
    alpha.mu = rep(0, 2),
    alpha.V = diag(1000, 2, 2)
  ))
)
```

```
mcmc_us <- MCMCglmm(cbind(speed_sc, exploration_sc) ~ trait - 1 +
  trait:assay_rep_sc +
  trait:body_size_sc,
  random = ~ us(trait):ID,
  rcov = ~ us(trait):units,
  family = c("gaussian", "gaussian"),
  prior = prior_1ex,
  nitt = 420000,
  burnin = 20000,
  thin = 100,
  verbose = FALSE,
  data = df_dragons
)
```

```
omar <- par()
par(mar = c(4, 2, 1.5, 2))
plot(mcmc_us$VCV[, c(1, 2, 4)])
```

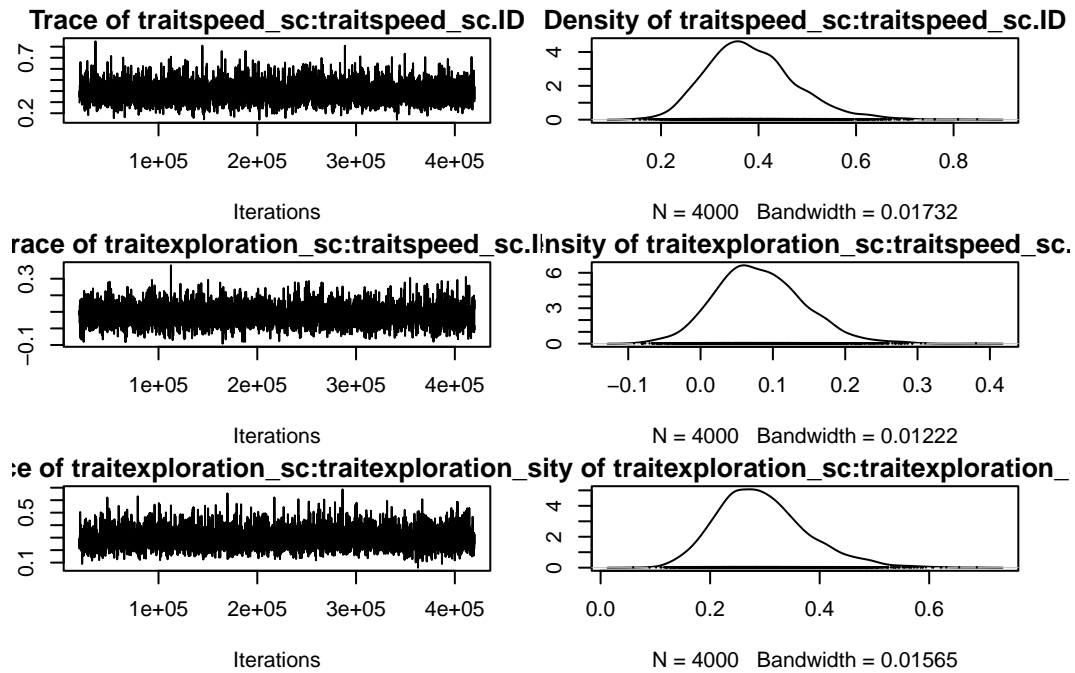


Figure 8.9.: MCMC trace and Posterior distribution of the (co)variance estimates of model mcmc_us

```
plot(mcmc_us$VCV[, c(5, 6, 8)])
```

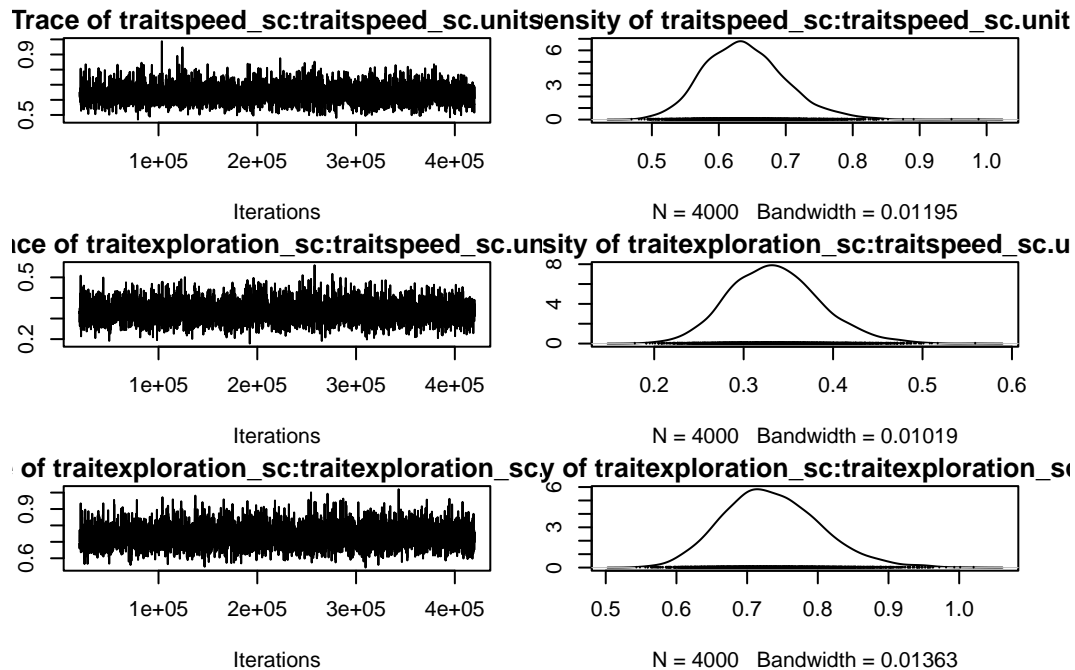


Figure 8.10.: MCMC trace and Posterior distribution of the (co)variance estimates of model mcmc_us

```
par(omar)
```

```
summary(mcmc_us)
```

```
Iterations = 20001:419901
Thinning interval = 100
Sample size = 4000
```

```
DIC: 1596.672
```

```
G-structure: ~us(trait):ID
```

| | post.mean | l-95% CI | u-95% CI | eff.samp |
|--|-----------|----------|----------|----------|
| traitspeed_sc:traitspeed_sc.ID | 0.3859 | 0.22108 | 0.5699 | 4000 |
| traitexploration_sc:traitspeed_sc.ID | 0.0818 | -0.02887 | 0.2085 | 4000 |
| traitspeed_sc:traitexploration_sc.ID | 0.0818 | -0.02887 | 0.2085 | 4000 |
| traitexploration_sc:traitexploration_sc.ID | 0.2952 | 0.14846 | 0.4666 | 4000 |

```
R-structure: ~us(trait):units
```

| | post.mean | l-95% CI | u-95% CI | eff.samp |
|---|-----------|----------|----------|----------|
| traitspeed_sc:traitspeed_sc.units | 0.6394 | 0.5273 | 0.7634 | |
| traitexploration_sc:traitspeed_sc.units | 0.3351 | 0.2404 | 0.4367 | |
| traitspeed_sc:traitexploration_sc.units | 0.3351 | 0.2404 | 0.4367 | |
| traitexploration_sc:traitexploration_sc.units | 0.7352 | 0.6058 | 0.8711 | |
| | eff.samp | | | |
| traitspeed_sc:traitspeed_sc.units | 3663 | | | |
| traitexploration_sc:traitspeed_sc.units | 4000 | | | |
| traitspeed_sc:traitexploration_sc.units | 4000 | | | |
| traitexploration_sc:traitexploration_sc.units | 3816 | | | |

```
Location effects: cbind(speed_sc, exploration_sc) ~ trait - 1 + trait:assay_rep_sc + trait:body_
```

| | post.mean | l-95% CI | u-95% CI | eff.samp |
|----------------------------------|------------|------------|-----------|----------|
| traitspeed_sc | 0.0006779 | -0.1518332 | 0.1753289 | 4000 |
| traitexploration_sc | -0.0020681 | -0.1512061 | 0.1486171 | 4000 |
| traitspeed_sc:assay_rep_sc | -0.0350510 | -0.1115105 | 0.0414211 | 4276 |
| traitexploration_sc:assay_rep_sc | -0.0210627 | -0.1033344 | 0.0677792 | 4184 |
| traitspeed_sc:body_size_sc | 0.1049471 | -0.0502591 | 0.2672082 | 4000 |
| traitexploration_sc:body_size_sc | 0.0734641 | -0.0813663 | 0.2167971 | 4000 |
| | pMCMC | | | |
| traitspeed_sc | 0.996 | | | |
| traitexploration_sc | 0.986 | | | |
| traitspeed_sc:assay_rep_sc | 0.377 | | | |
| traitexploration_sc:assay_rep_sc | 0.621 | | | |


```
traitspeed_sc:body_size_sc      0.199
traitexploration_sc:body_size_sc 0.346
```

```
mcmc_prop_f <- mcmc_us$VCV[, 1] /
  (mcmc_us$VCV[, 1] + mcmc_us$VCV[, 5])
plot(mcmc_prop_f)
```

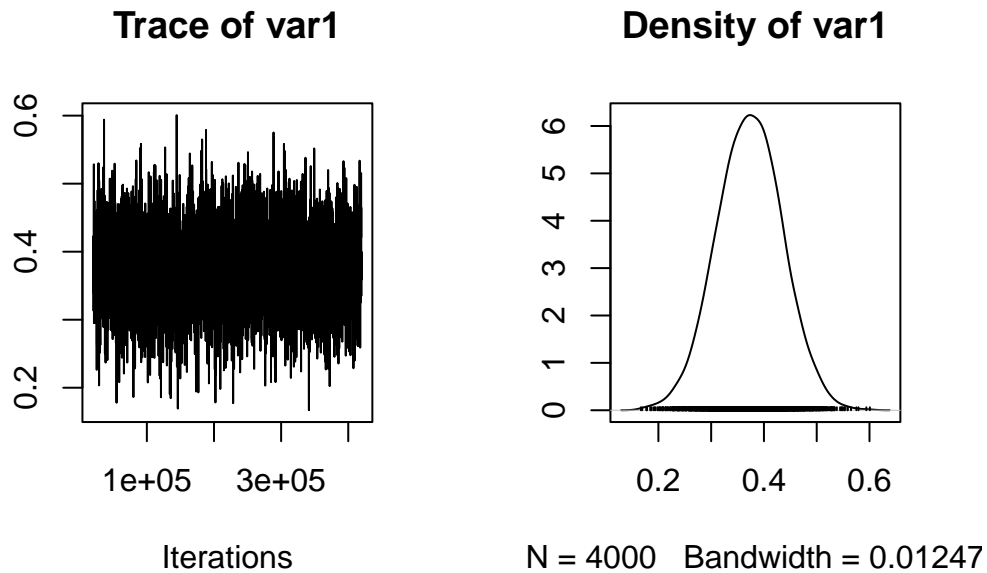


Figure 8.11.: Posterior trace and distribution of the repeatability in flying speed

```
posterior.mode(mcmc_prop_f)
```

```
var1
0.3704914
```

```
HPDinterval(mcmc_prop_f)
```

```
lower upper
var1 0.2514295 0.4929244
attr(,"Probability")
[1] 0.95
```

```
mcmc_prop_e <- mcmc_us$VCV[, 4] /
  (mcmc_us$VCV[, 4] + mcmc_us$VCV[, 8])
plot(mcmc_prop_e)
```

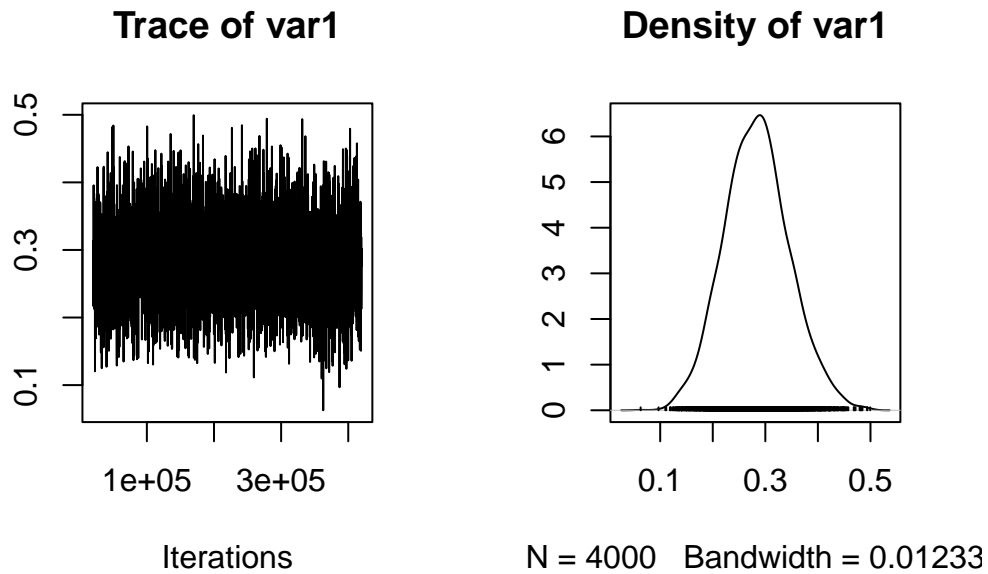


Figure 8.12.: Posterior trace and distribution of the repeatability of exploration

```
posterior.mode(mcmc_prop_e)
```

```
var1
0.2971507
```

```
HPDinterval(mcmc_prop_e)
```

```
lower upper
var1 0.16458 0.4103482
attr(,"Probability")
[1] 0.95
```

```
mcmc_cor_fe <- mcmc_us$VCV[, 2] /
  sqrt(mcmc_us$VCV[, 1] * mcmc_us$VCV[, 4])
plot(mcmc_cor_fe)
```

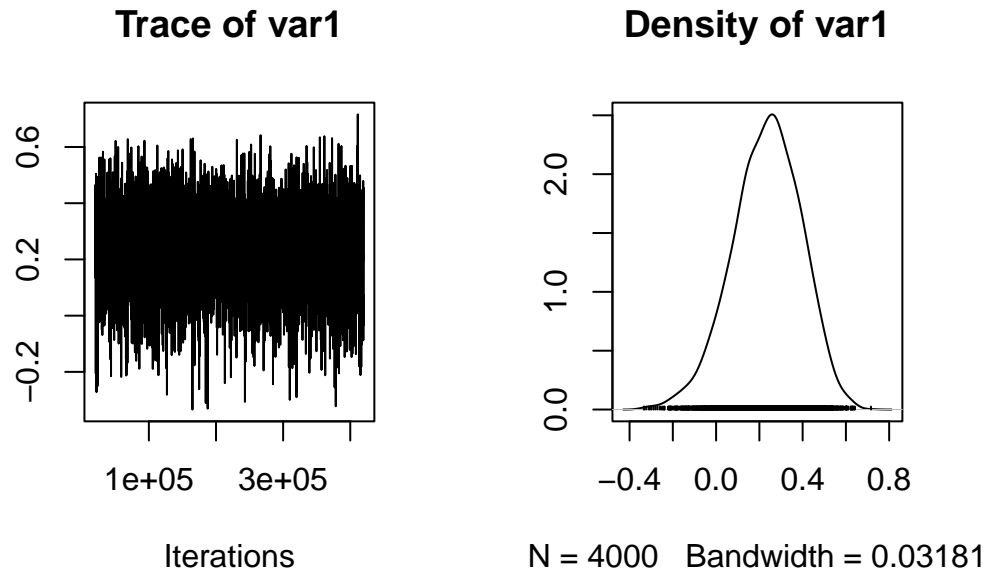


Figure 8.13.: Posterior trace and distribution of the correlation between flying speed and exploration

```
posterior.mode(mcmc_cor_fe)
```

```
var1
0.2775189
```

```
HPDinterval(mcmc_cor_fe)
```

```
      lower      upper
var1 -0.07038747 0.5358468
attr(,"Probability")
[1] 0.95
```

```
df_cor[3, 1] <- "MCMCglmm"
df_cor[3, -1] <- c(posterior.mode(mcmc_cor_fe), HPDinterval(mcmc_cor_fe))
rownames(df_cor) <- NULL

ggplot(df_cor, aes(x = Method, y = Correlation)) +
  geom_point() +
  geom_linerange(aes(ymin = low, ymax = high)) +
  ylim(-1, 1) +
  geom_hline(yintercept = 0, linetype = 2) +
  theme_classic()
```

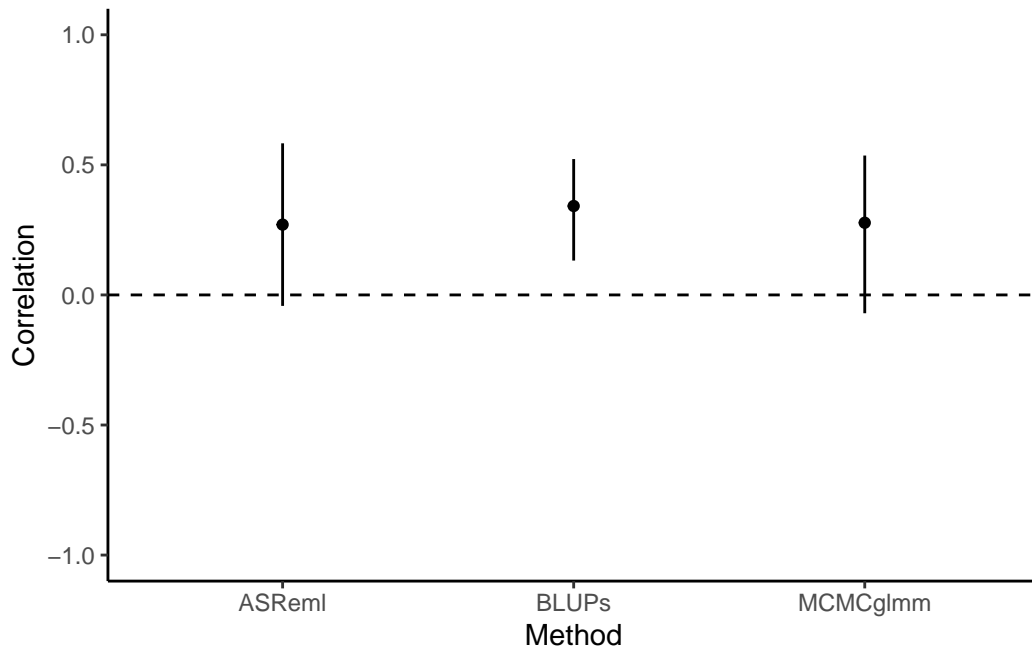


Figure 8.14.: Correlation estimates (with CI) using 3 different methods

Table 8.3.: Correlation (with 95% intervals) between flying speed and exploration estimated with 3 different methods

| Method | Correlation | low | high |
|----------|-------------|--------|-------|
| ASReml | 0.270 | -0.042 | 0.583 |
| BLUPs | 0.342 | 0.132 | 0.522 |
| MCMCglmm | 0.278 | -0.070 | 0.536 |

8.2.5. Happy multivariate models



Figure 8.15.: A female blue dragon of the West

9. Random regression and character state approaches

9.1. Lecture

Amazing beasties and crazy animals



Figure 9.1.: Dream pet dragon

9.2. Practical

In this practical, we will revisit our analysis on unicorn aggressivity. Honestly, we can use any other data with repeated measures for this exercise but I just ❤️ unicorns.

9.2.1. R packages needed

First we load required libraries

```
library(lme4)
library(tidyverse)
library(broom.mixed)
library(asreml)
library(MCMCglmm)
library(bayesplot)
```

9.2.2. Refresher on unicorn aggression

In the previous, practical on linear mixed models, we simply explored the differences among individuals in their mean aggression (Intercept), but we assumed that the response to the change in aggression with the opponent size (i.e. plasticity) was the same for all individuals. However, this plastic responses can also vary among individuals. This is called IxE, or individual by environment interaction. To test if individuals differ in their plasticity we can use a random regression, which is simply a mixed-model where we fit both a random intercept and a random slope effect.

Following analysis from the previous practical, our model of interest using scaled covariate was:

```
aggression ~ opp_size + body_size_sc + assay_rep_sc + block
            + (1 | ID)
```

We should start by loading the data and refitting the model using `lmer()`.

```
unicorns <- read.csv("data/unicorns_aggression.csv")
unicorns <- unicorns %>%
  mutate(
    body_size_sc = scale(body_size),
    assay_rep_sc = scale(assay_rep, scale = FALSE)
  )

m_mer <- lmer(
  aggression ~ opp_size + body_size_sc + assay_rep_sc + block
    + (1 | ID),
  data = unicorns
)
summary(m_mer)
```

```
Linear mixed model fit by REML. t-tests use Satterthwaite's method [
lmerModLmerTest]
Formula: aggression ~ opp_size + body_size_sc + assay_rep_sc + block +
(1 | ID)
Data: unicorns
```

```
REML criterion at convergence: 1136.5
```

```
Scaled residuals:
```

```
      Min       1Q   Median       3Q      Max
-2.85473 -0.62831  0.02545  0.68998  2.74064
```

```
Random effects:
```

```
Groups   Name             Variance Std.Dev.
ID       (Intercept)  0.02538  0.1593
Residual                   0.58048  0.7619
```

```
Number of obs: 480, groups: ID, 80
```


Fixed effects:

| | Estimate | Std. Error | df | t value | Pr(> t) |
|--------------|----------|------------|-----------|---------|------------|
| (Intercept) | 9.00181 | 0.03907 | 78.07315 | 230.395 | <2e-16 *** |
| opp_size | 1.05141 | 0.04281 | 396.99857 | 24.562 | <2e-16 *** |
| body_size_sc | 0.03310 | 0.03896 | 84.21144 | 0.850 | 0.398 |
| assay_rep_sc | -0.05783 | 0.04281 | 396.99857 | -1.351 | 0.177 |
| block | -0.02166 | 0.06955 | 397.00209 | -0.311 | 0.756 |

Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Correlation of Fixed Effects:

| | (Intr) | opp_sz | bdy_s_ | assy__ |
|-------------|--------|--------|--------|--------|
| opp_size | 0.000 | | | |
| body_siz_sc | 0.000 | 0.000 | | |
| assay_rp_sc | 0.000 | -0.100 | 0.000 | |
| block | 0.000 | 0.000 | 0.002 | 0.000 |

We can now plot the predictions for each of our observations and plot for the observed and the fitted data for each individuals. To do so we will use the `augment()` function from the  `broom.mixed`.

Below, we plot the raw data for each individual in one panel, with the fitted slopes in a second panel. Because we have 2 blocks of data, and `block` is fitted as a fixed effect, for ease of presentation we need to either select only 1 block for representation, take the average over the block effect or do a more complex graph with the two blocks. Here I have selected only one of the blocks for this plot

```
pred_m_mer <- augment(m_mer) %>%
  select(ID, block, opp_size, .fitted, aggression) %>%
  filter(block == -0.5) %>%
  gather(
    type, aggression,
    `\.fitted`:aggression
  )
ggplot(pred_m_mer, aes(x = opp_size, y = aggression, group = ID)) +
  geom_line(alpha = 0.3) +
  theme_classic() +
  facet_grid(. ~ type)
```

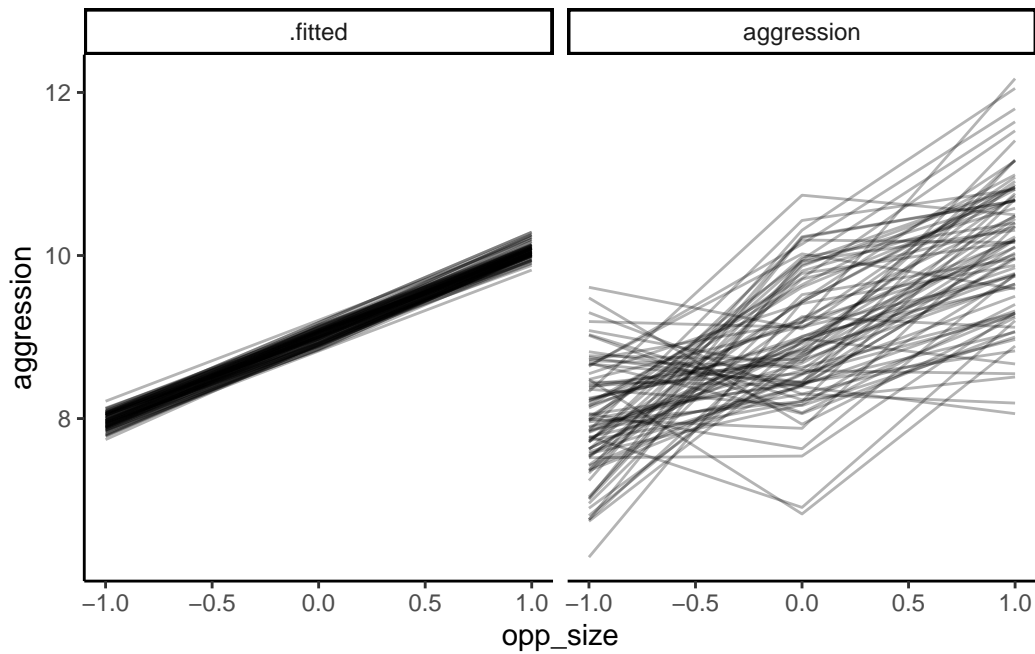


Figure 9.2.: Predicted (from `m_mer`) and observed value of aggression as a function of opponent size in unicorns

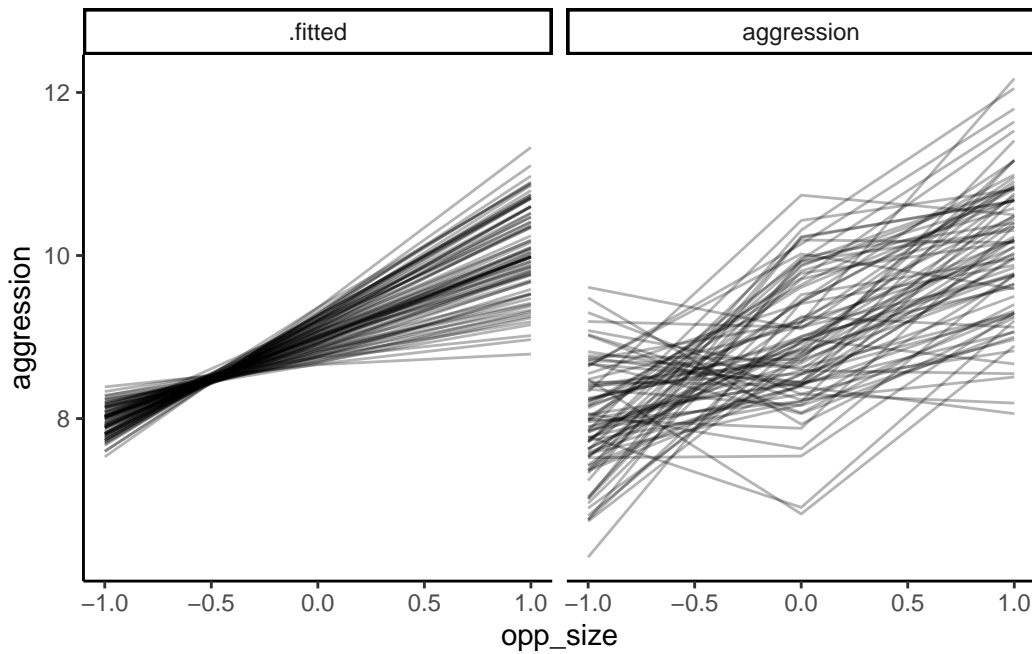
This illustrates the importance of using model predictions to see whether the model actually fits the individual-level data well or not — while the diagnostic plots looked fine, and the model captures mean plasticity, here we can see that the model really doesn't fit the actual data very well at all.

9.2.3. Random regression

9.2.3.1. with `lme4`

```
rr_mer <- lmer(
  aggression ~ opp_size + body_size_sc + assay_rep_sc + block
  + (1 + opp_size | ID),
  data = unicorns
)
```

```
pred_rr_mer <- augment(rr_mer) %>%
  select(ID, block, opp_size, .fitted, aggression) %>%
  filter(block == -0.5) %>%
  gather(type, aggression, `.`.fitted`.aggression)
ggplot(pred_rr_mer, aes(x = opp_size, y = aggression, group = ID)) +
  geom_line(alpha = 0.3) +
  theme_classic() +
  facet_grid(. ~ type)
```

We can test the improvement of the model fit using the overloaded `anova` function in R to perform a likelihood ratio test (LRT):

```
anova(rr_mer, m_mer, refit = FALSE)
```

| | npar | AIC | BIC | logLik | deviance | Chisq | Df | Pr(>Chisq) |
|--------|------|----------|----------|-----------|----------|---------|----|------------|
| m_mer | 7 | 1150.477 | 1179.693 | -568.2383 | 1136.477 | NA | NA | NA |
| rr_mer | 9 | 1092.356 | 1129.920 | -537.1780 | 1074.356 | 62.1206 | 2 | 0 |

We can see here that the LRT uses a chi-square test with 2 degrees of freedom, and indicates that the random slopes model shows a statistically significant improvement in model fit. The 2df are because there are two additional (co)variance terms estimated in the random regression model: a variance term for individual slopes, and the covariance (or correlation) between the slopes and intercepts. Let's look at those values, and also the fixed effects parameters, via the model summary:

```
summary(rr_mer)
```

```
Linear mixed model fit by REML. t-tests use Satterthwaite's method [
lmerModLmerTest]
Formula: aggression ~ opp_size + body_size_sc + assay_rep_sc + block +
  (1 + opp_size | ID)
Data: unicorns
```

```
REML criterion at convergence: 1074.4
```

Scaled residuals:

| Min | 1Q | Median | 3Q | Max |
|----------|----------|----------|---------|---------|
| -3.04932 | -0.59780 | -0.02002 | 0.59574 | 2.68010 |

Random effects:

| Groups | Name | Variance | Std.Dev. | Corr |
|----------|-------------|----------|----------|------|
| ID | (Intercept) | 0.05043 | 0.2246 | |
| | opp_size | 0.19167 | 0.4378 | 0.96 |
| Residual | | 0.42816 | 0.6543 | |

Number of obs: 480, groups: ID, 80

Fixed effects:

| | Estimate | Std. Error | df | t value | Pr(> t) |
|--------------|----------|------------|-----------|---------|------------|
| (Intercept) | 9.00181 | 0.03902 | 78.44088 | 230.707 | <2e-16 *** |
| opp_size | 1.05033 | 0.06123 | 79.50694 | 17.153 | <2e-16 *** |
| body_size_sc | 0.02725 | 0.03377 | 84.34959 | 0.807 | 0.422 |
| assay_rep_sc | -0.04702 | 0.03945 | 387.69415 | -1.192 | 0.234 |
| block | -0.02169 | 0.05973 | 318.19553 | -0.363 | 0.717 |

Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Correlation of Fixed Effects:

| | (Intr) | opp_sz | bdy_s_ | assy__ |
|-------------|--------|--------|--------|--------|
| opp_size | 0.495 | | | |
| body_siz_sc | 0.000 | 0.000 | | |
| assay_rp_sc | 0.000 | -0.064 | -0.006 | |
| block | 0.000 | 0.000 | 0.002 | 0.000 |

9.2.3.2. with asreml

```
unicorns <- unicorns %>%
  mutate( ID = as.factor(ID))
rr_asr <- asreml(
  aggression ~ opp_size + body_size_sc + assay_rep_sc + block,
  random = ~str(~ ID + ID:opp_size, ~us(2):id(ID)),
  residual = ~ units,
  data = unicorns,
  maxiter = 200
)
```

Model fitted using the gamma parameterization.

ASReml 4.1.0 Thu Feb 1 15:44:20 2024

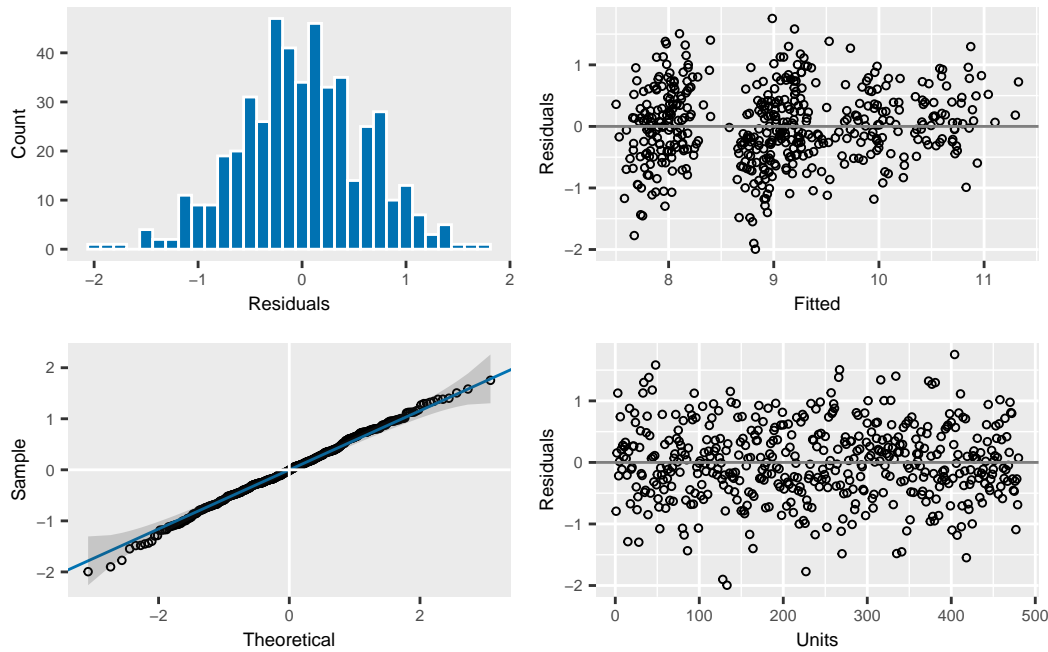
| | LogLik | Sigma2 | DF | wall | cpu |
|---|----------|----------|-----|----------|-----|
| 1 | -109.426 | 0.463232 | 475 | 15:44:20 | 0.0 |
| 2 | -105.050 | 0.454593 | 475 | 15:44:20 | 0.0 |
| 3 | -101.814 | 0.443662 | 475 | 15:44:20 | 0.0 |

```

4      -100.814      0.433873      475 15:44:20      0.0
5      -100.683      0.428596      475 15:44:20      0.0
6      -100.682      0.428170      475 15:44:20      0.0

```

```
plot(rr_asr)
```



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

```

              solution  std error    z.ratio
block          -0.02168725  0.05973354  -0.3630665
assay_rep_sc  -0.04702032  0.03944594  -1.1920191
body_size_sc   0.02725092  0.03377443   0.8068506
opp_size       1.05032703  0.06123110  17.1534907
(Intercept)    9.00181250  0.03901766 230.7112239

```

```
wa <- wald(rr_asr, ssType = "conditional", denDF = "numeric")
```

Model fitted using the gamma parameterization.

ASReml 4.1.0 Thu Feb 1 15:44:20 2024

```

      LogLik      Sigma2      DF      wall      cpu
1      -100.682      0.428168      475 15:44:20      0.0
2      -100.682      0.428168      475 15:44:20      0.0

```

Calculating denominator DF

```
attr(wa$Wald, "heading") <- NULL
wa
```

```
$Wald
```

| | Df | denDF | F.inc | F.con | Margin | Pr |
|--------------|----|-------|-------|-------|--------|---------|
| (Intercept) | 1 | 78.3 | 65490 | 53230 | | 0.00000 |
| opp_size | 1 | 79.5 | 293 | 294 | A | 0.00000 |
| body_size_sc | 1 | 84.3 | 1 | 1 | A | 0.42202 |
| assay_rep_sc | 1 | 387.6 | 1 | 1 | A | 0.23398 |
| block | 1 | 318.1 | 0 | 0 | A | 0.71680 |

```
$stratumVariances
```

| | df | Variance | ID+ID:opp_size!us(2)_1:1 |
|--------------------------|-----------|-----------|---|
| ID+ID:opp_size!us(2)_1:1 | 78.00483 | 0.4790737 | 5.216311 |
| ID+ID:opp_size!us(2)_2:1 | 0.00000 | 0.0000000 | 0.000000 |
| ID+ID:opp_size!us(2)_2:2 | 78.94046 | 1.1937287 | 0.000000 |
| units!R | 318.05470 | 0.4281680 | 0.000000 |
| | | | ID+ID:opp_size!us(2)_2:1 ID+ID:opp_size!us(2)_2:2 |
| ID+ID:opp_size!us(2)_1:1 | | -3.301137 | 0.5221955 |
| ID+ID:opp_size!us(2)_2:1 | | 0.000000 | 0.000000 |
| ID+ID:opp_size!us(2)_2:2 | | 0.000000 | 3.9943993 |
| units!R | | 0.000000 | 0.000000 |
| | | | units!R |
| ID+ID:opp_size!us(2)_1:1 | | 1 | |
| ID+ID:opp_size!us(2)_2:1 | | 1 | |
| ID+ID:opp_size!us(2)_2:2 | | 1 | |
| units!R | | 1 | |

```
summary(rr_asr)$varcomp
```

| | component | std.error | z.ratio | bound | %ch |
|--------------------------|------------|------------|-----------|-------|-----|
| ID+ID:opp_size!us(2)_1:1 | 0.05042932 | 0.02027564 | 2.487187 | P | 0 |
| ID+ID:opp_size!us(2)_2:1 | 0.09458336 | 0.02400745 | 3.939751 | P | 0 |
| ID+ID:opp_size!us(2)_2:2 | 0.19165924 | 0.04832059 | 3.966409 | P | 0 |
| units!R | 0.42816954 | 0.03395320 | 12.610582 | P | 0 |

```
rio_asr <- asreml(
  aggression ~ opp_size + body_size_sc + assay_rep_sc + block,
  random = ~ ID,
  residual = ~units,
  data = unicorns,
  maxiter = 200
)
```

Model fitted using the gamma parameterization.

ASReml 4.1.0 Thu Feb 1 15:44:20 2024

| | LogLik | Sigma2 | DF | wall | cpu |
|---|----------|----------|-----|----------|-----|
| 1 | -132.611 | 0.560353 | 475 | 15:44:21 | 0.0 |
| 2 | -132.106 | 0.567043 | 475 | 15:44:21 | 0.0 |
| 3 | -131.796 | 0.575157 | 475 | 15:44:21 | 0.0 |
| 4 | -131.743 | 0.580762 | 475 | 15:44:21 | 0.0 |
| 5 | -131.742 | 0.580480 | 475 | 15:44:21 | 0.0 |

```
pchisq(2 * (rr_asr$loglik - rio_asr$loglik), 2,
  lower.tail = FALSE
)
```

```
[1] 3.241026e-14
```

```
vpredict(rr_asr, cor_is ~ V2 / (sqrt(V1) * sqrt(V3)))
```

| | Estimate | SE |
|--------|-----------|-----------|
| cor_is | 0.9620736 | 0.1773965 |

```
pred_rr_asr <- as.data.frame(predict(rr_asr,
  classify = "opp_size:ID",
  levels = list(
    "opp_size" =
      c(opp_size = -1:1)
  )
)$pvals)
```

Model fitted using the gamma parameterization.

ASReml 4.1.0 Thu Feb 1 15:44:21 2024

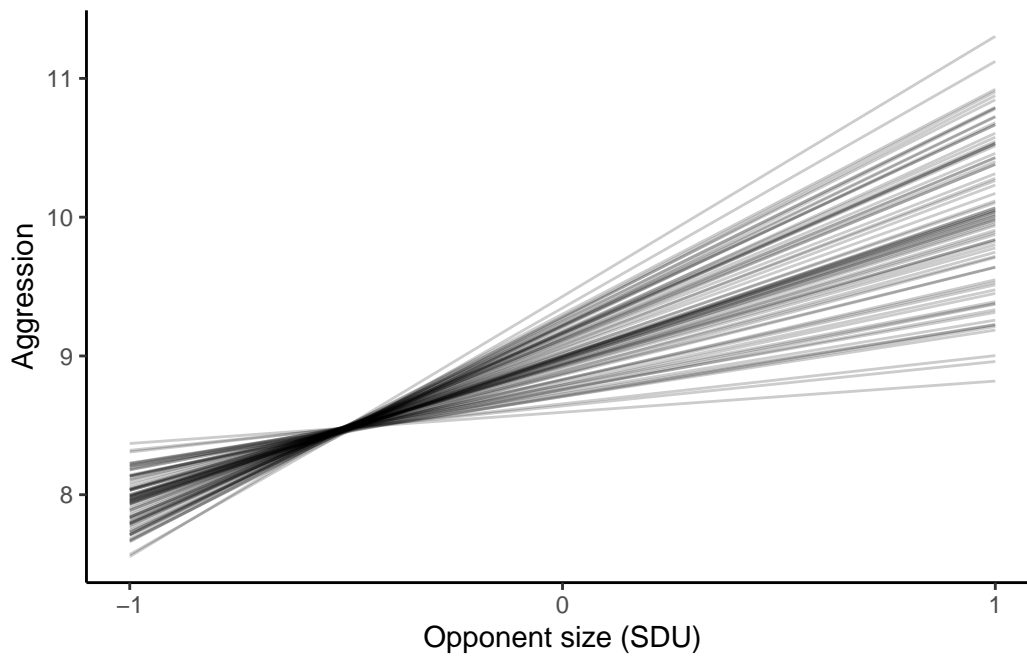
| | LogLik | Sigma2 | DF | wall | cpu |
|---|----------|----------|-----|----------|-----|
| 1 | -100.682 | 0.428168 | 475 | 15:44:21 | 0.1 |
| 2 | -100.682 | 0.428168 | 475 | 15:44:21 | 0.0 |
| 3 | -100.682 | 0.428168 | 475 | 15:44:21 | 0.0 |

```
p_rr <- ggplot(pred_rr_asr, aes(x = opp_size,
  y = predicted.value,
  group = ID)) +
  geom_line(alpha = 0.2) +
```

```

scale_x_continuous(breaks = c(-1, 0, 1)) +
  labs(
    x = "Opponent size (SDU)",
    y = "Aggression"
  ) +
  theme_classic()
p_rr

```



9.2.3.3. with MCMCglmm

```

prior_RR <- list(
  R = list(V = 1, nu = 0.002),
  G = list(
    G1 = list(V = diag(2)*0.02, nu = 3,
alpha.mu = rep(0, 2),
alpha.V= diag(1000, 2, 2))))
rr_mcmc <- MCMCglmm(
  aggression ~ opp_size + assay_rep_sc + body_size_sc + block,
  random = ~ us(1 + opp_size):ID,
  rcov = ~ units,
  family = "gaussian",
  prior = prior_RR,
  nitt=750000,
  burnin=50000,
  thin=350,

```

```

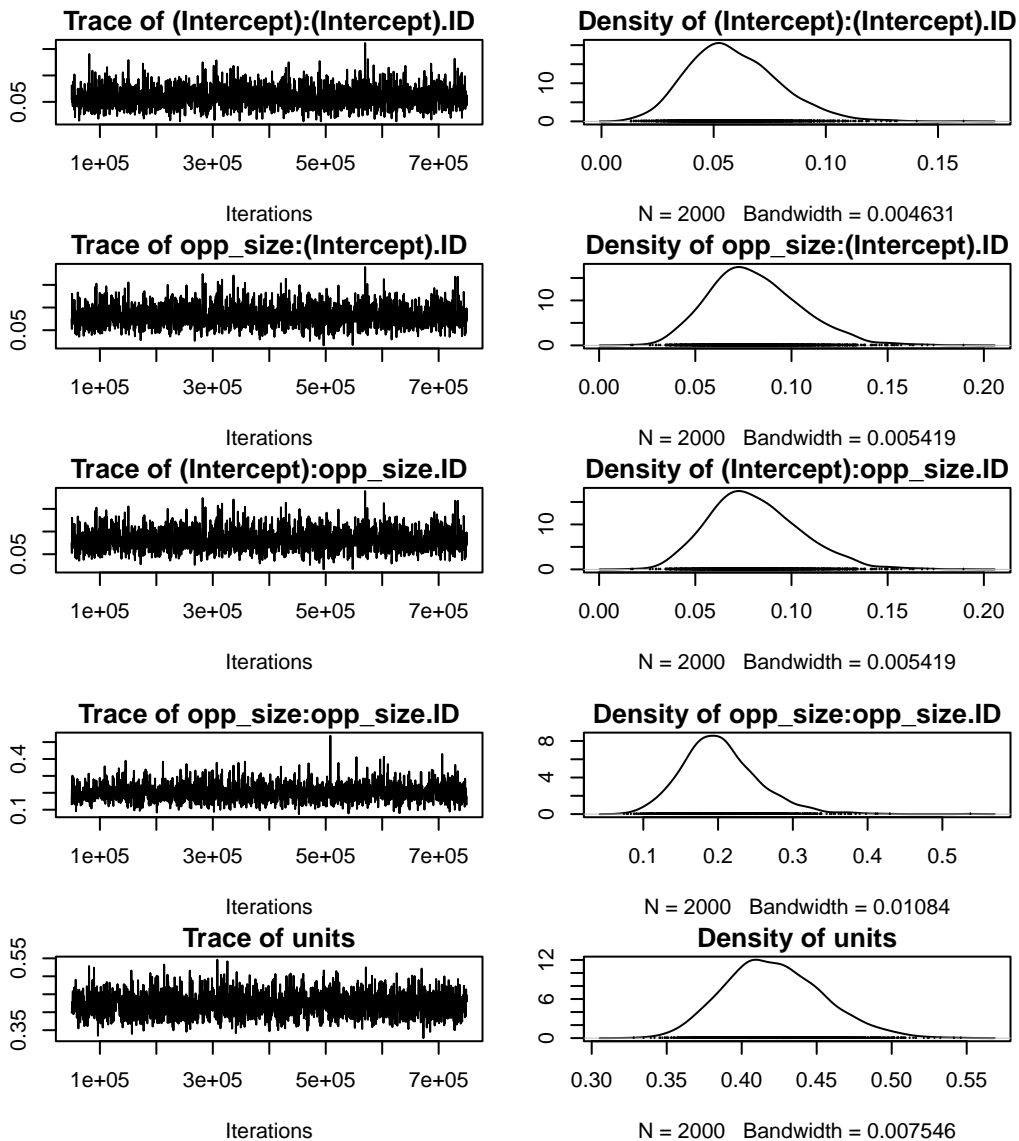
verbose = FALSE,
data = unicorns,
pr = TRUE,
saveX = TRUE, saveZ = TRUE)

```

```

omar <- par()
par(mar = c(4, 2, 1.5, 2))
plot(rr_mcmc$VCV)

```



```
par(omar)
```

```
Warning in par(omar): graphical parameter "cin" cannot be set
```

```
Warning in par(omar): graphical parameter "cra" cannot be set
```

```
Warning in par(omar): graphical parameter "csi" cannot be set
```

```
Warning in par(omar): graphical parameter "cxy" cannot be set
```

```
Warning in par(omar): graphical parameter "din" cannot be set
```

```
Warning in par(omar): graphical parameter "page" cannot be set
```

```
posterior.mode(rr_mcmc$VCV[, "opp_size:opp_size.ID"]) # mean
```

```
var1
0.174148
```

```
HPDinterval(rr_mcmc$VCV[, "opp_size:opp_size.ID"])
```

```
      lower      upper
var1 0.1092903 0.3067066
attr(,"Probability")
[1] 0.95
```

```
rr_cor_mcmc <- rr_mcmc$VCV[, "opp_size:(Intercept).ID"] /
  (sqrt(rr_mcmc$VCV[, "(Intercept):(Intercept).ID"]) *
   sqrt(rr_mcmc$VCV[, "opp_size:opp_size.ID"]))
posterior.mode(rr_cor_mcmc)
```

```
var1
0.8305619
```



```
HPDinterval(rr_cor_mcmc)
```

```
      lower      upper
var1 0.5098244 0.9797991
attr(,"Probability")
[1] 0.95
```

```
df_rand <- cbind(unicorns,
  rr_fit = predict(rr_mcmc, marginal = NULL)
) %>%
  select(ID, opp_size, rr_fit, aggression) %>%
  group_by(ID, opp_size) %>%
  summarise(
    rr_fit = mean(rr_fit),
    aggression = mean(aggression)
) %>%
  gather(
    Type, Value,
    rr_fit:aggression
  )
```

``summarise()`` has grouped output by 'ID'. You can override using the ``.groups`` argument.

```
# Plot separate panels for individual lines of each type
ggplot(df_rand, aes(x = opp_size, y = Value, group = ID)) +
  geom_line(alpha = 0.3) +
  scale_x_continuous(breaks = c(-1, 0, 1)) +
  theme_classic() +
  facet_grid(. ~ Type)
```

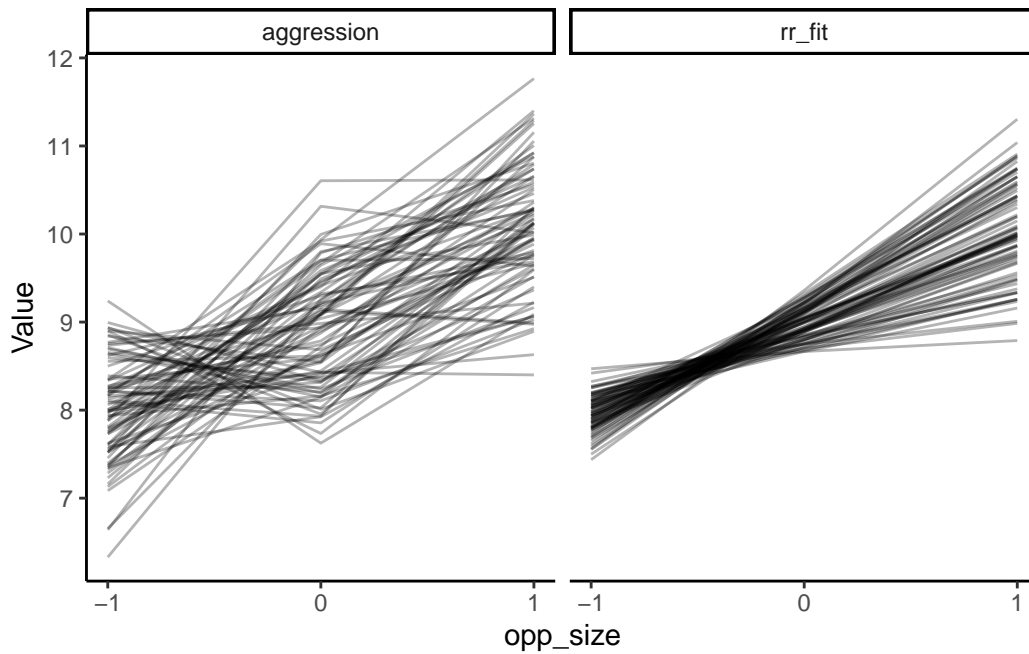


Table 9.2.: Variance estimated from random regression models using 3 different softwares

| Method | v_int | cov | v_sl | v_r |
|----------|-----------|-----------|-----------|-----------|
| lmer | 0.0504347 | 0.0945863 | 0.1916653 | 0.4281625 |
| asreml | 0.0504293 | 0.0945834 | 0.1916592 | 0.4281695 |
| MCMCglmm | 0.0511152 | 0.0725605 | 0.1741480 | 0.4046411 |

9.2.4. Character-State approach

Need to pivot to a wider format

```
unicorns_cs <- unicorns %>%
  select(ID, body_size, assay_rep, block, aggression, opp_size) %>%
  mutate(
    opp_size = recode(as.character(opp_size), "-1" = "s", "0" = "m", "1" = "l")
  ) %>%
  dplyr::rename(agg = aggression) %>%
  pivot_wider(names_from = opp_size, values_from = c(agg, assay_rep)) %>%
  mutate(
    body_size_sc = scale(body_size),
    opp_order = as.factor(paste(assay_rep_s, assay_rep_m, assay_rep_l, sep = "_"))
  )
str(unicorns_cs)
```

```
tibble [160 x 11] (S3: tbl_df/tbl/data.frame)
 $ ID          : Factor w/ 80 levels "ID_1","ID_10",...: 1 1 2 2 3 3 4 4 5 5 ...
 $ body_size   : num [1:160] 206 207 283 288 229 ...
```

```

$ block      : num [1:160] -0.5 0.5 -0.5 0.5 -0.5 0.5 -0.5 0.5 -0.5 0.5 ...
$ agg_s      : num [1:160] 7.02 8.44 7.73 8.08 8.06 8.16 8.16 8.51 7.59 6.67 ...
$ agg_l      : num [1:160] 10.67 10.51 10.81 10.67 9.77 ...
$ agg_m      : num [1:160] 10.22 8.95 9.43 9.46 7.63 ...
$ assay_rep_s : int [1:160] 1 3 2 2 1 1 3 3 1 1 ...
$ assay_rep_l : int [1:160] 2 2 1 1 2 2 2 1 2 2 ...
$ assay_rep_m : int [1:160] 3 1 3 3 3 3 1 2 3 3 ...
$ body_size_sc: num [1:160, 1] -1.504 -1.456 0.988 1.143 -0.76 ...
  ..- attr(*, "scaled:center")= num 253
  ..- attr(*, "scaled:scale")= num 31.1
$ opp_order   : Factor w/ 6 levels "1_2_3","1_3_2",...: 2 5 4 4 2 2 5 6 2 2 ...

```

```
head(unicorns_cs)
```

```

# A tibble: 6 x 11
  ID      body_size block agg_s agg_l agg_m assay_rep_s assay_rep_l assay_rep_m
  <fct>    <dbl> <dbl> <dbl> <dbl> <dbl>      <int>      <int>      <int>
1 ID_1      206.  -0.5  7.02 10.7 10.2          1          2          3
2 ID_1      207.   0.5  8.44 10.5  8.95          3          2          1
3 ID_10     283.  -0.5  7.73 10.8  9.43          2          1          3
4 ID_10     288.   0.5  8.08 10.7  9.46          2          1          3
5 ID_11     229.  -0.5  8.06  9.77  7.63          1          2          3
6 ID_11     236.   0.5  8.16 10.8  8.23          1          2          3
# i 2 more variables: body_size_sc <dbl[,1]>, opp_order <fct>

```

```

cs_asr <- asreml(
  cbind(agg_s, agg_m, agg_l) ~ trait + trait:body_size_sc +
  trait:block +
  trait:opp_order,
  random =~ ID:us(trait),
  residual =~ units:us(trait),
  data = unicorns_cs,
  maxiter = 200
)

```

Model fitted using the sigma parameterization.

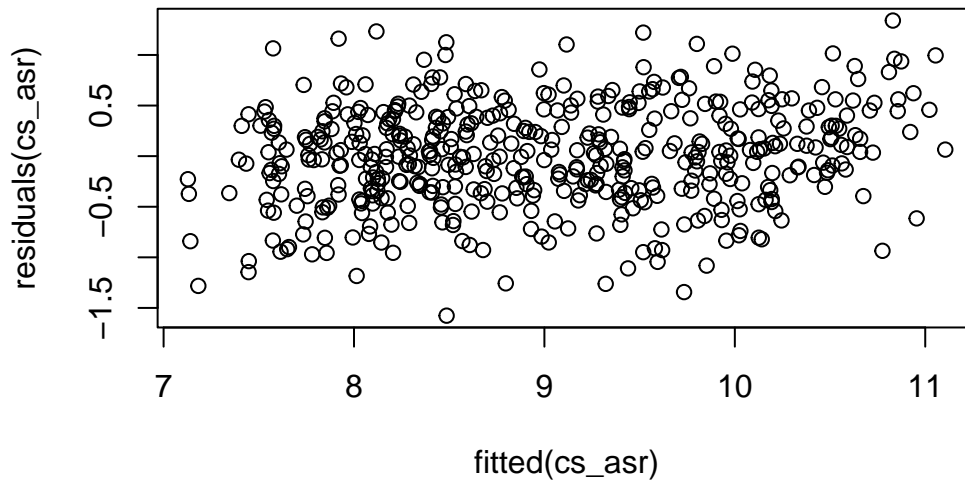
```

ASReml 4.1.0 Thu Feb 1 15:46:46 2024

```

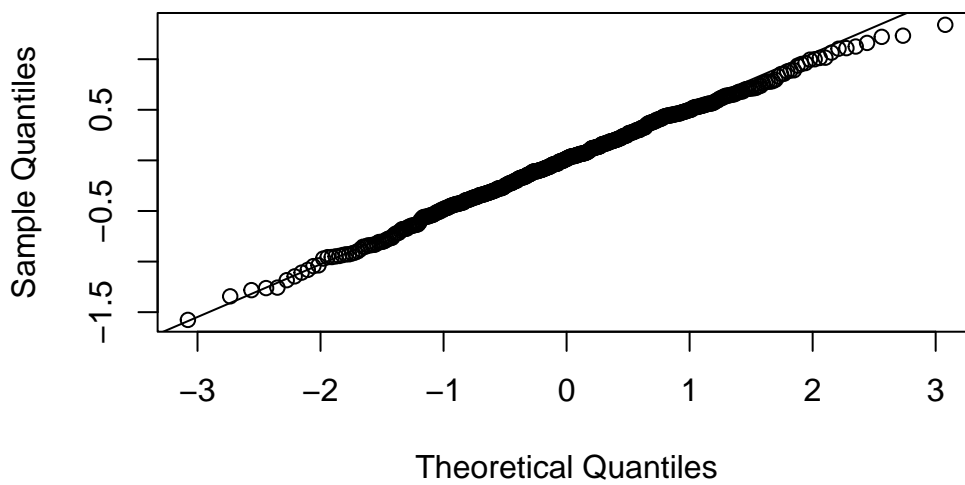
| | LogLik | Sigma2 | DF | wall | cpu |
|---|----------|--------|-----|----------|-----|
| 1 | -150.172 | 1.0 | 456 | 15:46:46 | 0.0 |
| 2 | -129.658 | 1.0 | 456 | 15:46:46 | 0.0 |
| 3 | -110.454 | 1.0 | 456 | 15:46:46 | 0.0 |
| 4 | -101.879 | 1.0 | 456 | 15:46:46 | 0.0 |
| 5 | -100.092 | 1.0 | 456 | 15:46:46 | 0.0 |
| 6 | -100.054 | 1.0 | 456 | 15:46:46 | 0.0 |
| 7 | -100.054 | 1.0 | 456 | 15:46:46 | 0.0 |

```
plot(residuals(cs_asr) ~ fitted(cs_asr))
```

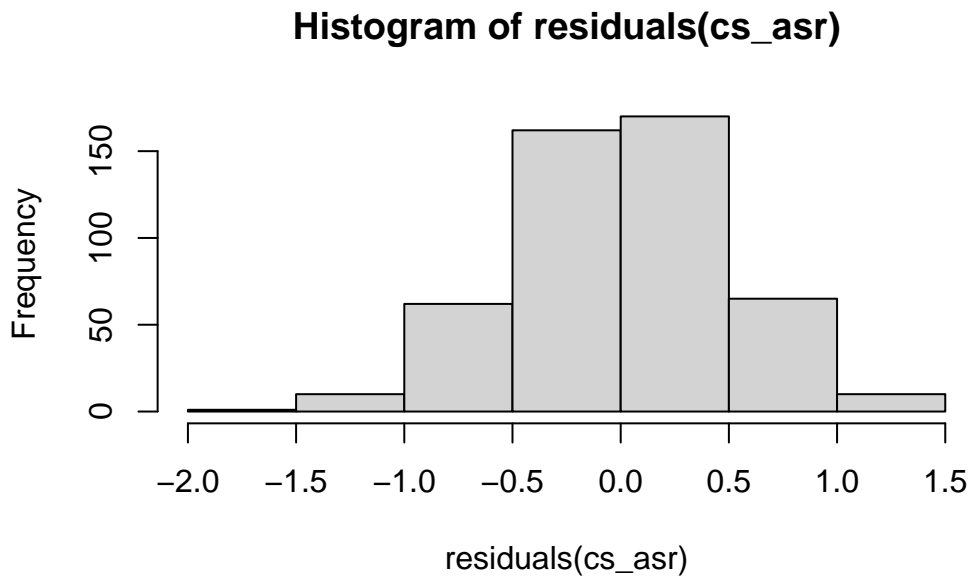


```
qqnorm(residuals(cs_asr))  
qqline(residuals(cs_asr))
```

Normal Q-Q Plot



```
hist(residuals(cs_asr))
```



```
summary(cs_asr, all = T)$coef.fixed
```

NULL

```
wa <- wald(cs_asr, ssType = "conditional", denDF = "numeric")
```

Model fitted using the sigma parameterization.

ASReml 4.1.0 Thu Feb 1 15:46:47 2024

| | LogLik | Sigma2 | DF | wall | cpu |
|---|----------|--------|-----|----------|-----|
| 1 | -100.054 | 1.0 | 456 | 15:46:47 | 0.0 |
| 2 | -100.054 | 1.0 | 456 | 15:46:47 | 0.0 |

Calculating denominator DF

```
attr(wa$Wald, "heading") <- NULL
wa
```

\$Wald

| | Df | denDF | F.inc | F.con | Margin | Pr |
|--|----|-------|-------|-------|--------|----|
|--|----|-------|-------|-------|--------|----|

```

trait          3  73.2 21080.0 21080.0      0.00000
trait:body_size_sc 3  86.6   0.4   0.5      B 0.68324
trait:block    3  75.2   0.6   0.3      B 0.82418
trait:opp_order 15 240.5   1.3   1.3      B 0.23282

```

```
$stratumVariances
```

```
NULL
```

```
summary(cs_asr)$varcomp[, c("component", "std.error")]
```

```

                                component  std.error
ID:trait!trait_agg_s:agg_s      0.192959991 0.06321872
ID:trait!trait_agg_m:agg_s     -0.168519644 0.05085583
ID:trait!trait_agg_m:agg_m      0.245594370 0.07096325
ID:trait!trait_agg_l:agg_s     -0.151990204 0.05660748
ID:trait!trait_agg_l:agg_m      0.158418588 0.06374995
ID:trait!trait_agg_l:agg_l      0.312548090 0.09125168
units:trait!R                   1.000000000      NA
units:trait!trait_agg_s:agg_s   0.318089965 0.05198135
units:trait!trait_agg_m:agg_s   0.010362390 0.03695483
units:trait!trait_agg_m:agg_m   0.322379911 0.05248291
units:trait!trait_agg_l:agg_s  -0.009311656 0.04168455
units:trait!trait_agg_l:agg_m   0.159240476 0.04569305
units:trait!trait_agg_l:agg_l   0.405942147 0.06679700

```

```

cs_idh_asr <- asreml(
  cbind(agg_s, agg_m, agg_l) ~ trait + trait:body_size_sc +
  trait:block +
  trait:opp_order,
  random = ~ ID:idh(trait),
  residual = ~ units:us(trait),
  data = unicorns_cs,
  maxiter = 200
)

```

Model fitted using the sigma parameterization.

```
ASReml 4.1.0 Thu Feb 1 15:46:47 2024
```

| | LogLik | Sigma2 | DF | wall | cpu |
|---|----------|--------|-----|----------|-----|
| 1 | -147.068 | 1.0 | 456 | 15:46:47 | 0.0 |
| 2 | -131.268 | 1.0 | 456 | 15:46:47 | 0.0 |
| 3 | -116.908 | 1.0 | 456 | 15:46:47 | 0.0 |
| 4 | -110.996 | 1.0 | 456 | 15:46:47 | 0.0 |
| 5 | -109.905 | 1.0 | 456 | 15:46:47 | 0.0 |
| 6 | -109.866 | 1.0 | 456 | 15:46:47 | 0.0 |
| 7 | -109.863 | 1.0 | 456 | 15:46:47 | 0.0 |

```
pchisq(2 * (cs_asr$loglik - cs_idh_asr$loglik), 3,
      lower.tail = FALSE
    )
```

```
[1] 0.0002038324
```

```
vpredict(cs_asr, cor_S_M ~ V2 / (sqrt(V1) * sqrt(V3)))
```

| | Estimate | SE |
|---------|------------|-----------|
| cor_S_M | -0.7741189 | 0.1869789 |

```
vpredict(cs_asr, cor_M_L ~ V5 / (sqrt(V3) * sqrt(V6)))
```

| | Estimate | SE |
|---------|-----------|-----------|
| cor_M_L | 0.5717926 | 0.1469504 |

```
vpredict(cs_asr, cor_S_L ~ V4 / (sqrt(V1) * sqrt(V6)))
```

| | Estimate | SE |
|---------|------------|-----------|
| cor_S_L | -0.6189044 | 0.1912133 |

```
vpredict(cs_asr, prop_S ~ V1 / (V1 + V8))
```

| | Estimate | SE |
|--------|-----------|------------|
| prop_S | 0.3775756 | 0.09950306 |

```
vpredict(cs_asr, prop_M ~ V3 / (V3 + V10))
```

| | Estimate | SE |
|--------|----------|-----------|
| prop_M | 0.432404 | 0.0934477 |

```
vpredict(cs_asr, prop_L ~ V6 / (V6 + V13))
```

```

      Estimate      SE
prop_L 0.4350067 0.09498512

```

```

init_CS_cor1_tri <- c(
  0.999,
  0.999, 0.999,
  1, 1, 1
)
names(init_CS_cor1_tri) <- c(
  "F",
  "F", "F",
  "U", "U", "U"
)
cs_asr_cor1_tri <- asreml(
  cbind(agg_s, agg_m, agg_l) ~ trait + trait:body_size_sc +
  trait:block +
  trait:opp_order,
  random = ~ ID:corgh(trait, init = init_CS_cor1_tri),
  residual = ~ units:us(trait),
  data = unicorns_cs,
  maxiter = 500
)

```

Model fitted using the sigma parameterization.

ASReml 4.1.0 Thu Feb 1 15:46:47 2024

| | LogLik | Sigma2 | DF | wall | cpu |
|----|----------|--------|-----|----------|--------------------|
| 1 | -228.016 | 1.0 | 456 | 15:46:47 | 0.0 (3 restrained) |
| 2 | -150.014 | 1.0 | 456 | 15:46:47 | 0.0 |
| 3 | -129.580 | 1.0 | 456 | 15:46:47 | 0.0 |
| 4 | -119.992 | 1.0 | 456 | 15:46:47 | 0.0 (1 restrained) |
| 5 | -116.907 | 1.0 | 456 | 15:46:47 | 0.0 (1 restrained) |
| 6 | -115.772 | 1.0 | 456 | 15:46:47 | 0.0 |
| 7 | -115.647 | 1.0 | 456 | 15:46:47 | 0.0 |
| 8 | -115.588 | 1.0 | 456 | 15:46:47 | 0.0 |
| 9 | -115.533 | 1.0 | 456 | 15:46:47 | 0.0 |
| 10 | -115.479 | 1.0 | 456 | 15:46:47 | 0.0 |
| 11 | -115.427 | 1.0 | 456 | 15:46:47 | 0.0 |
| 12 | -115.378 | 1.0 | 456 | 15:46:47 | 0.0 |
| 13 | -115.331 | 1.0 | 456 | 15:46:47 | 0.0 |
| 14 | -115.289 | 1.0 | 456 | 15:46:47 | 0.0 |
| 15 | -115.251 | 1.0 | 456 | 15:46:47 | 0.0 |
| 16 | -115.217 | 1.0 | 456 | 15:46:47 | 0.0 |
| 17 | -115.188 | 1.0 | 456 | 15:46:47 | 0.0 |

| | | | | | |
|----|----------|-----|-----|----------|--------------------|
| 18 | -115.162 | 1.0 | 456 | 15:46:47 | 0.0 |
| 19 | -115.141 | 1.0 | 456 | 15:46:47 | 0.0 |
| 20 | -115.122 | 1.0 | 456 | 15:46:47 | 0.0 |
| 21 | -115.107 | 1.0 | 456 | 15:46:47 | 0.0 |
| 22 | -115.093 | 1.0 | 456 | 15:46:47 | 0.0 |
| 23 | -115.082 | 1.0 | 456 | 15:46:47 | 0.0 |
| 24 | -115.073 | 1.0 | 456 | 15:46:47 | 0.0 (1 restrained) |
| 25 | -115.064 | 1.0 | 456 | 15:46:47 | 0.0 |
| 26 | -115.064 | 1.0 | 456 | 15:46:47 | 0.0 |

```
pchisq(2 * (cs_asr$loglik - cs_asr_cor1_tri$loglik),
3,
  lower.tail = FALSE
)
```

```
[1] 1.367792e-06
```

```
df_CS_pred <- as.data.frame(predict(cs_asr,
  classify = "trait:ID"
)$pvals)
```

Model fitted using the sigma parameterization.

ASReml 4.1.0 Thu Feb 1 15:46:47 2024

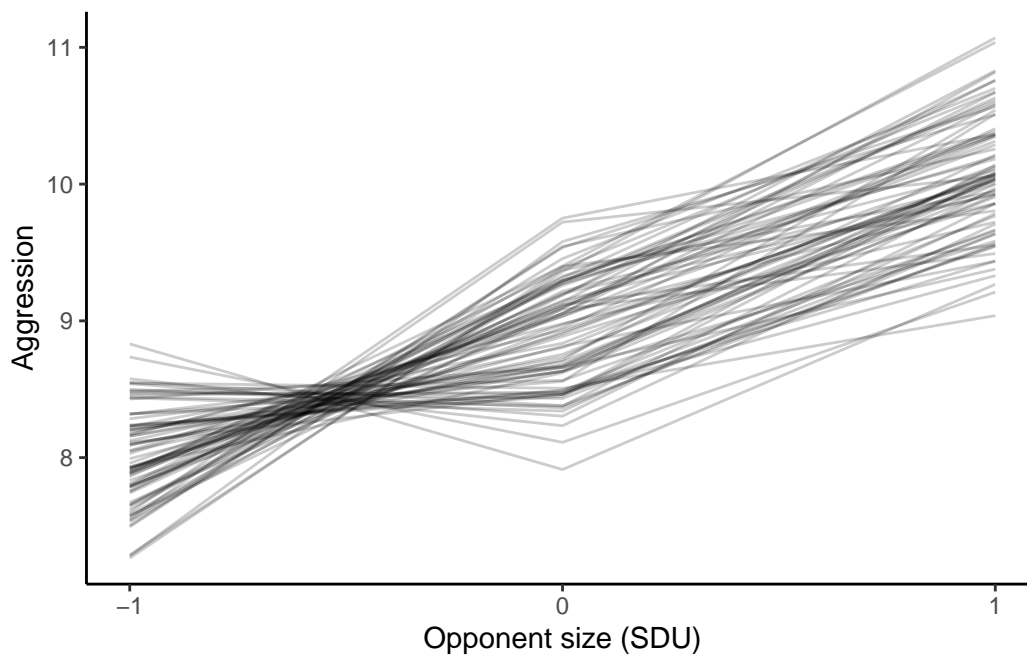
| | LogLik | Sigma2 | DF | wall | cpu |
|---|----------|--------|-----|----------|-----|
| 1 | -100.054 | 1.0 | 456 | 15:46:47 | 0.1 |
| 2 | -100.054 | 1.0 | 456 | 15:46:47 | 0.0 |
| 3 | -100.054 | 1.0 | 456 | 15:46:47 | 0.0 |

```
# Add numeric variable for easier plotting
# of opponent size
df_CS_pred <- df_CS_pred %>%
  mutate(sizeNum = ifelse(trait == "agg_s", -1,
    ifelse(trait == "agg_m", 0, 1)
  ))
p_cs <- ggplot(df_CS_pred, aes(
  x = sizeNum,
  y = predicted.value,
  group = ID
)) +
  geom_line(alpha = 0.2) +
  scale_x_continuous(breaks = c(-1, 0, 1)) +
  labs(
```

```

  x = "Opponent size (SDU)",
  y = "Aggression"
) +
theme_classic()
p_cs

```



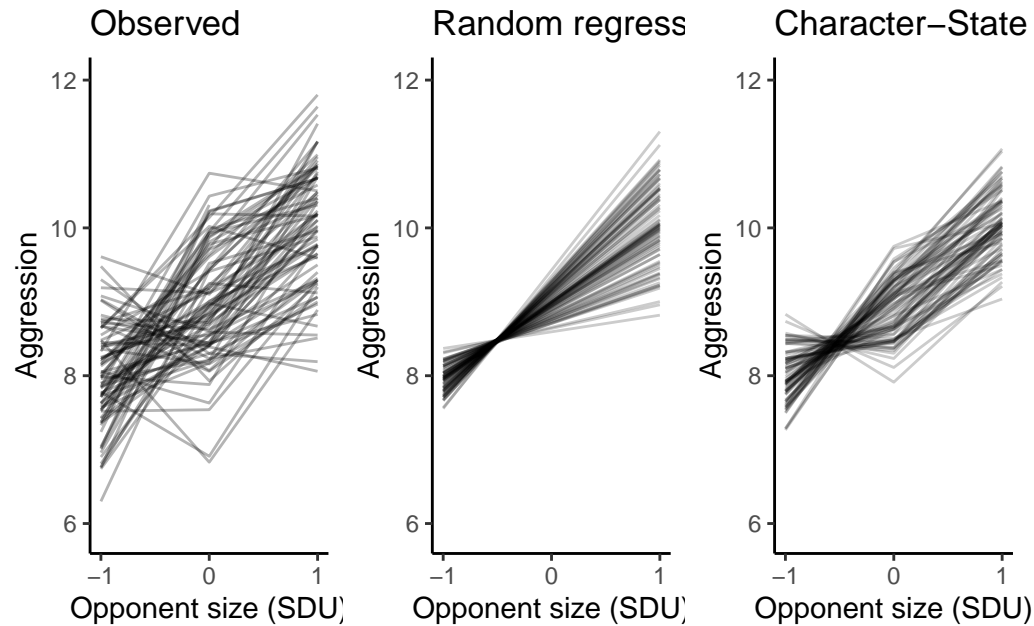
```

unicorns <- arrange(unicorns, opp_size, by_group = ID)
p_obs <- ggplot(unicorns[unicorns$block==-0.5,], aes(x = opp_size, y = aggression, group = ID)) +
  geom_line(alpha = 0.3) +
  scale_x_continuous(breaks = c(-1, 0, 1)) +
  labs(
    x = "Opponent size (SDU)",
    y = "Aggression"
  ) +
  ggtitle("Observed") +
  ylim(5.9, 12) +
  theme_classic()

p_rr <- p_rr + ggtitle("Random regression") + ylim(5.9, 12)
p_cs <- p_cs + ggtitle("Character-State") + ylim(5.9, 12)
p_obs + p_rr + p_cs

```

Warning: Removed 2 rows containing missing values (`geom_line()`).



9.2.5. From random regression to character-state

```
var_mat_asr <- function(model, var_names, pos){
  size <- length(var_names)
  v_out <- matrix(NA, ncol = size, nrow = size)
  rownames(v_out) <- var_names
  colnames(v_out) <- var_names
  v_out[upper.tri(v_out, diag = TRUE)] <- summary(model)$varcomp[pos, 1]
  v_out <- forceSymmetric(v_out, uplo = "U")
  as.matrix(v_out)
}
v_id_rr <- var_mat_asr(rr_asr, c("v_int", "v_sl"), 1:3)
knitr::kable(v_id_rr, digits = 3)
```

| | v_int | v_sl |
|-------|-------|-------|
| v_int | 0.050 | 0.095 |
| v_sl | 0.095 | 0.192 |

```
v_id_cs <- var_mat_asr(cs_asr, c("v_s", "v_m", "v_l"), 1:6)
knitr::kable(v_id_cs, digits = 3)
```

| | v_s | v_m | v_l |
|-----|--------|--------|--------|
| v_s | 0.193 | -0.169 | -0.152 |
| v_m | -0.169 | 0.246 | 0.158 |
| v_l | -0.152 | 0.158 | 0.313 |

We also need to make a second matrix, let's call it \mathbf{Q} (no particular reason, pick something else if you want). This is going to contain the values needed to turn an individual's intercept (mean) and slope (plasticity) deviations into estimates of environment-specific individual merit in a character state model.

What do we mean by this? Well if an individual i has an intercept deviation of $ID_{int(i)}$ and a slope deviation of $ID_{slp(i)}$ for a given value of the environment `opp_size` we might be interested in:

$$ID_i = (1 \times ID_{int(i)}) + (opp_size \times ID_{slp(i)})$$

We want to look at character states representing the three observed values of `opp_size` here so

```
Q <- as.matrix(cbind(c(1, 1, 1),
                    c(-1, 0, 1)))
```

Then we can generate our among-individual covariance matrix environment specific aggressiveness, which we can call `ID_cs_rr` by matrix multiplication:

```
ID_cs_rr <- Q %*% v_id_rr %*% t(Q) #where t(Q) is the transpose of Q
#and %*% is matrix multiplication

ID_cs_rr #rows and columns correspond to aggressiveness at opp_size=-1,0,1 in that order
```

```
      [,1]      [,2]      [,3]
[1,] 0.05292184 -0.04415404 -0.1412299
[2,] -0.04415404 0.05042932 0.1450127
[3,] -0.14122993 0.14501267 0.4312553
```

```
cov2cor(ID_cs_rr) #Converting to a correlation scale
```

```
      [,1]      [,2]      [,3]
[1,] 1.0000000 -0.8546956 -0.9348503
[2,] -0.8546956 1.0000000 0.9833253
[3,] -0.9348503 0.9833253 1.0000000
```

```
cov2cor(v_id_cs)
```

```
      v_s      v_m      v_l
v_s 1.0000000 -0.7741189 -0.6189044
v_m -0.7741189 1.0000000 0.5717926
v_l -0.6189044 0.5717926 1.0000000
```

9.2.6. Conclusions

9.2.7. Happy multivariate models



Figure 9.3.: A female blue dragon of the West

10. Beyond $P < 0.05$

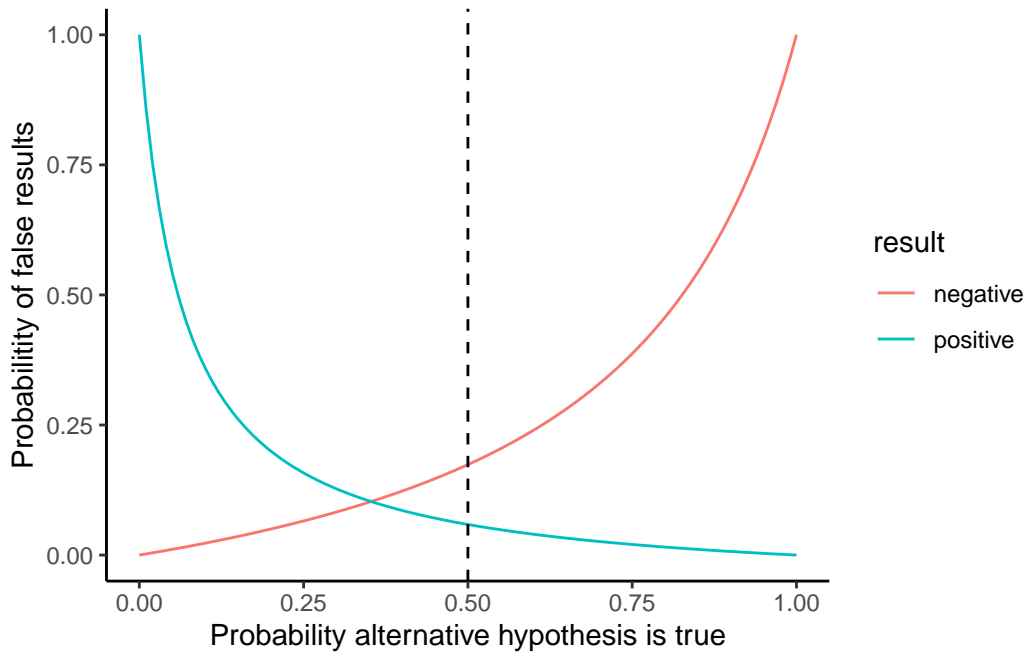
cite a bunch a must read paper on the subject and maybe summarize the big point of **Do** and **Don't**

```
library(ggplot2)

alpha <- 0.05
beta <- 0.2

p_h1_true <- seq(0, 1, length = 100)
p_fp <- alpha * (1 - p_h1_true) /
  (alpha * (1 - p_h1_true) + (1 - beta) * p_h1_true)
p_fn <- beta * p_h1_true /
  (beta * p_h1_true + (1 - alpha) * (1 - p_h1_true))

dat <- rbind(
  data.frame(p_h1 = p_h1_true, prob = p_fp, result = "positive" ),
  data.frame(p_h1 = p_h1_true, prob = p_fn, result = "negative")
)
ggplot(dat, aes(x = p_h1, y = prob, colour = result)) +
  geom_line() +
  geom_vline(xintercept = 0.5, linetype = 2) +
  xlab("Probability alternative hypothesis is true") +
  ylab("Probability of false results") +
  xlim(0, 1) +
  theme_classic()
```



References

R packages

This book was produced using all the following R packages

| Package | Version | Citation |
|----------------|------------|---|
| asreml | 4.2.0.302 | The VSNi Team (2023) |
| base | 4.3.3 | R Core Team (2024) |
| bayesplot | 1.11.1 | Gabry et al. (2019); Gabry and Mahr (2024) |
| bookdown | 0.38 | Xie (2016); Xie (2024a) |
| brms | 2.21.0 | Bürkner (2017); Bürkner (2018); Bürkner (2021) |
| broom.mixed | 0.2.9.5 | Bolker and Robinson (2024) |
| DHARMA | 0.4.6 | Hartig (2022) |
| emoji | 15.0 | Hvitfeldt (2022) |
| factoextra | 1.0.7 | Kassambara and Mundt (2020) |
| FactoMineR | 2.10 | Lê et al. (2008) |
| grateful | 0.2.4 | Francisco Rodriguez-Sanchez and Connor P. Jackson (2023) |
| knitr | 1.45 | Xie (2014); Xie (2015); Xie (2023) |
| lattice | 0.22.6 | Sarkar (2008) |
| lme4 | 1.1.35.2 | Bates et al. (2015) |
| lmerTest | 3.1.3 | Kuznetsova et al. (2017) |
| MASS | 7.3.60.0.1 | Venables and Ripley (2002) |
| MCMCglmm | 2.35 | Hadfield (2010) |
| mvtnorm | 1.2.4 | Genz and Bretz (2009) |
| nadiv | 2.17.3 | Wolak (2012) |
| palmerpenguins | 0.1.1 | Horst et al. (2020) |
| patchwork | 1.2.0 | Pedersen (2024) |
| performance | 0.11.0 | Lüdecke et al. (2021) |
| rmarkdown | 2.26 | Xie et al. (2018); Xie et al. (2020); Allaire et al. (2024) |
| rptR | 0.9.22 | Stoffel et al. (2017) |
| tidybayes | 3.0.6 | Kay (2023) |
| tidyverse | 2.0.0 | Wickham et al. (2019) |

| Package | Version | Citation |
|--------------|---------|-----------------------------|
| abind | 1.4.5 | Plate and Heiberger (2016) |
| ape | 5.7.1 | Paradis and Schliep (2019) |
| arrayhelpers | 1.1.0 | Beleites (2020) |
| askpass | 1.2.0 | Ooms (2023a) |
| backports | 1.4.1 | Lang and R Core Team (2021) |
| base64enc | 0.1.3 | Urbanek (2015) |

References

| Package | Version | Citation |
|----------------|----------|--|
| bayestestR | 0.13.2 | Makowski et al. (2019) |
| BH | 1.84.0.0 | Eddelbuettel et al. (2024a) |
| bit | 4.0.5 | Oehlschlägel and Ripley (2022) |
| bit64 | 4.0.5 | Oehlschlägel and Silvestri (2020) |
| blob | 1.2.4 | Wickham (2023a) |
| bridgesampling | 1.1.2 | Gronau et al. (2020) |
| brio | 1.1.4 | Hester and Csárdi (2023) |
| Brodbingnag | 1.2.9 | Hankin (2007) |
| bslib | 0.7.0 | Sievert et al. (2024) |
| cachem | 1.0.8 | Chang (2023a) |
| callr | 3.7.6 | Csárdi and Chang (2024a) |
| car | 3.1.2 | Fox and Weisberg (2019) |
| carData | 3.0.5 | Fox et al. (2022) |
| cellranger | 1.1.0 | Bryan (2016) |
| checkmate | 2.3.1 | Lang (2017) |
| clipr | 0.8.0 | Lincoln (2022) |
| coda | 0.19.4.1 | Plummer et al. (2006) |
| colorspace | 2.1.0 | Zeileis et al. (2009); Stauffer et al. (2009); Zeileis et al. (2020) |
| commonmark | 1.9.1 | Ooms (2024a) |
| corpcor | 1.6.10 | Schafer et al. (2021) |
| corrplot | 0.92 | Wei and Simko (2021) |
| cowplot | 1.1.3 | Wilke (2024a) |
| cpp11 | 0.4.7 | Vaughan et al. (2023) |
| crayon | 1.5.2 | Csárdi (2022) |
| crosstalk | 1.2.1 | Cheng and Sievert (2023) |
| cubature | 2.1.0 | Narasimhan et al. (2023) |
| curl | 5.2.1 | Ooms (2024b) |
| data.table | 1.15.4 | Barrett et al. (2024) |
| datawizard | 0.10.0 | Patil et al. (2022) |
| DBI | 1.2.2 | R Special Interest Group on Databases (R-SIG-DB) et al. (2024) |
| dendextend | 1.17.1 | Galili (2015) |
| desc | 1.4.3 | Csárdi et al. (2023) |
| diffobj | 0.3.5 | Gaslam (2021) |
| digest | 0.6.35 | Antoine Lucas et al. (2024) |
| distributional | 0.4.0 | O'Hara-Wild et al. (2024) |
| doParallel | 1.0.17 | Corporation and Weston (2022) |
| DT | 0.32 | Xie et al. (2024) |
| ellipse | 0.5.0 | Murdoch and Chow (2023) |
| ellipsis | 0.3.2 | Wickham (2021) |
| emmeans | 1.10.0 | Lenth (2024b) |
| estimability | 1.5 | Lenth (2024a) |
| evaluate | 0.23 | Wickham and Xie (2023) |
| fansi | 1.0.6 | Gaslam (2023) |
| farver | 2.1.1 | Pedersen et al. (2022) |
| fastmap | 1.1.1 | Chang (2023b) |
| flashClust | 1.1.2 | Langfelder and Horvath (2012) |
| fontawesome | 0.5.2 | Iannone (2023) |
| foreach | 1.5.2 | Microsoft and Weston (2022) |

| Package | Version | Citation |
|--------------|------------|---|
| fs | 1.6.3 | Hester et al. (2023b) |
| furrr | 0.3.1 | Vaughan and Dancho (2022) |
| future | 1.33.2 | @ |
| future.apply | 1.11.2 | @ |
| gap | 1.5.3 | Zhao (2007); Zhao (2023a) |
| gap.datasets | 0.0.6 | Zhao (2023b) |
| gargle | 1.5.2 | Bryan et al. (2023) |
| generics | 0.1.3 | Wickham et al. (2022a) |
| ggdist | 3.3.2 | Kay (2024a); Kay (2024b) |
| ggpubr | 0.6.0 | Kassambara (2023a) |
| ggrepel | 0.9.5 | Slowikowski (2024) |
| ggridges | 0.5.6 | Wilke (2024b) |
| ggsci | 3.0.3 | Xiao (2024) |
| ggsignif | 0.6.4 | Constantin and Patil (2021) |
| globals | 0.16.3 | Bengtsson (2024a) |
| glue | 1.7.0 | Hester and Bryan (2024) |
| gridExtra | 2.3 | Auguie (2017) |
| gtable | 0.3.4 | Wickham and Pedersen (2023) |
| highr | 0.10 | Xie and Qiu (2022) |
| htmltools | 0.5.8 | Cheng et al. (2024c) |
| htmlwidgets | 1.6.4 | Vaidyanathan et al. (2023) |
| httpuv | 1.6.15 | Cheng et al. (2024a) |
| ids | 1.0.1 | FitzJohn (2017) |
| inline | 0.3.19 | Sklyar et al. (2021) |
| insight | 0.19.10 | Lüdecke et al. (2019) |
| isoband | 0.2.7 | Wickham et al. (2022b) |
| iterators | 1.0.14 | Analytics and Weston (2022) |
| jquerylib | 0.1.4 | Sievert and Cheng (2021) |
| labeling | 0.4.3 | Justin Talbot (2023) |
| later | 1.3.2 | Chang and Cheng (2023) |
| lazyeval | 0.2.2 | Wickham (2019) |
| leaps | 3.1 | Fortran code by Alan Miller (2020) |
| lifecycle | 1.0.4 | Henry and Wickham (2023) |
| listenv | 0.9.1 | Bengtsson (2024b) |
| lmtest | 0.9.40 | Zeileis and Hothorn (2002) |
| loo | 2.7.0 | Vehtari et al. (2017); Yao et al. (2017); Vehtari et al. (2024) |
| MatrixModels | 0.5.3 | Bates and Maechler (2023) |
| matrixStats | 1.2.0 | Bengtsson (2023) |
| memoise | 2.0.1 | Wickham et al. (2021) |
| mime | 0.12 | Xie (2021) |
| minqa | 1.2.6 | Bates et al. (2023) |
| multcompView | 0.1.10 | Graves et al. (2024) |
| munsell | 0.5.0 | Wickham (2018) |
| nleqslv | 3.3.5 | Hasselmann (2023) |
| nloptr | 2.0.3 | Johnson (?) |
| numDeriv | 2016.8.1.1 | Gilbert and Varadhan (2019) |
| openssl | 2.1.1 | Ooms (2023b) |
| parallelly | 1.37.1 | Bengtsson (2024c) |

References

| Package | Version | Citation |
|---------------|-----------|---|
| pbapply | 1.7.2 | Solymos and Zawadzki (2023) |
| pbkrtest | 0.5.2 | Halekoh and Højsgaard (2014) |
| pkgbuild | 1.4.4 | Wickham et al. (2024b) |
| pkgconfig | 2.0.3 | Csárdi (2019) |
| pkgload | 1.3.4 | Wickham et al. (2024a) |
| plotly | 4.10.4 | Sievert (2020) |
| plyr | 1.8.9 | Wickham (2011a) |
| polynom | 1.4.1 | Venables et al. (2022) |
| posterior | 1.5.0 | Vehtari et al. (2021); Bürkner et al. (2023) |
| praise | 1.0.0 | Csardi and Sorhus (2015) |
| prettyunits | 1.2.0 | Csardi (2023a) |
| processx | 3.8.4 | Csárdi and Chang (2024b) |
| progress | 1.2.3 | Csárdi and FitzJohn (2023) |
| promises | 1.2.1 | Cheng (2023) |
| ps | 1.7.6 | Loden et al. (2024) |
| qgam | 1.3.4 | Fasiolo et al. (2021) |
| quadprog | 1.5.8 | Berwin A. Turlach R port by Andreas Weingessel
<Andreas.Weingessel@ci.tuwien.ac.at> Fortran contributions from Cleve Moler dpodi/LINPACK) (2019) |
| quantreg | 5.97 | Koenker (2023) |
| QuickJSR | 1.1.3 | Johnson (2024) |
| R6 | 2.5.1 | Chang (2021) |
| rappdirs | 0.3.3 | Ratnakumar et al. (2021) |
| rbibutils | 2.2.16 | Boshnakov and Putman (2023) |
| RColorBrewer | 1.1.3 | Neuwirth (2022) |
| Rcpp | 1.0.12 | Eddelbuettel and François (2011); Eddelbuettel (2013); Eddelbuettel and Balamuta (2018); Eddelbuettel et al. (2024b) |
| RcppEigen | 0.3.4.0.0 | Bates and Eddelbuettel (2013) |
| RcppParallel | 5.1.7 | Allaire et al. (2023) |
| Rdpack | 2.6 | Boshnakov (2023) |
| rematch | 2.0.0 | Csardi (2023b) |
| rematch2 | 2.1.2 | Csárdi (2020) |
| remotes | 2.5.0 | Csárdi et al. (2024) |
| renv | 1.0.5 | Ushey and Wickham (2024) |
| reshape2 | 1.4.4 | Wickham (2007) |
| rprojroot | 2.0.4 | Müller (2023) |
| rstan | 2.32.6 | Stan Development Team (2024) |
| rstantools | 2.4.0 | Gabry et al. (2024) |
| rstatix | 0.7.2 | Kassambara (2023b) |
| sass | 0.4.9 | Cheng et al. (2024b) |
| scales | 1.3.0 | Wickham et al. (2023b) |
| scatterplot3d | 0.3.44 | Ligges and Mächler (2003) |
| selectr | 0.4.2 | Potter (2012) |
| shiny | 1.8.1 | Chang et al. (2024) |
| sourcetools | 0.1.7.1 | Ushey (2023) |
| SparseM | 1.81 | Koenker (2021) |
| StanHeaders | 2.32.6 | Stan Development Team (2020) |
| stringi | 1.8.3 | Gagolewski (2022) |

| Package | Version | Citation |
|-------------|----------|---------------------------------|
| svUnit | 1.0.6 | Grosjean (2024) |
| sys | 3.4.2 | Ooms (2023c) |
| systemfonts | 1.0.6 | Pedersen et al. (2024) |
| tensorA | 0.36.2.1 | van den Boogaart (2023) |
| testthat | 3.2.1 | Wickham (2011b) |
| textshaping | 0.3.7 | Pedersen (2023) |
| tidyselect | 1.2.1 | Henry and Wickham (2024) |
| timechange | 0.3.0 | Spinu (2024) |
| tinytex | 0.50 | Xie (2019); Xie (2024b) |
| tzdb | 0.4.0 | Vaughan (2023) |
| utf8 | 1.2.4 | Perry (2023) |
| uuid | 1.2.0 | Urbanek and Ts'o (2024) |
| vctrs | 0.6.5 | Wickham et al. (2023a) |
| viridis | 0.6.5 | Garnier et al. (2024) |
| viridisLite | 0.4.2 | Garnier et al. (2023) |
| vroom | 1.6.5 | Hester et al. (2023a) |
| waldo | 0.5.2 | Wickham (2023b) |
| withr | 3.0.0 | Hester et al. (2024) |
| xfun | 0.43 | Xie (2024c) |
| xtable | 1.8.4 | Dahl et al. (2019) |
| yaml | 2.3.8 | Garbett et al. (2023) |
| zoo | 1.8.12 | Zeileis and Grothendieck (2005) |

Bibliography

- Allaire, J., R. Francois, K. Ushey, G. Vandenbrouck, M. Geelnard, and Intel. 2023. RcppParallel: Parallel programming tools for “Rcpp”.
- Allaire, J., Y. Xie, C. Dervieux, J. McPherson, J. Luraschi, K. Ushey, A. Atkins, H. Wickham, J. Cheng, W. Chang, and R. Iannone. 2024. rmarkdown: Dynamic documents for r.
- Analytics, R., and S. Weston. 2022. iterators: Provides iterator construct.
- Antoine Lucas, D. E. with contributions by, J. Tuszynski, H. Bengtsson, S. Urbanek, M. Frasca, B. Lewis, M. Stokely, H. Muehleisen, D. Murdoch, J. Hester, W. Wu, Q. Kou, T. Onkelinx, M. Lang, V. Simko, K. Hornik, R. Neal, K. Bell, M. de Queljoe, I. Suruceanu, B. Denney, D. Schumacher, W. Chang, D. Attali, and M. Chirico. 2024. digest: Create compact hash digests of r objects.
- Auguie, B. 2017. gridExtra: Miscellaneous functions for “Grid” graphics.
- Banta, J. A., M. H. H. Stevens, and M. Pigliucci. 2010. A comprehensive test of the “limiting resources” framework applied to plant tolerance to apical meristem damage. *Oikos* 119:359–369.
- Barrett, T., M. Dowle, A. Srinivasan, J. Gorecki, M. Chirico, and T. Hocking. 2024. data.table: Extension of “data.frame”.
- Bates, D., and D. Eddelbuettel. 2013. Fast and elegant numerical linear algebra using the RcppEigen package. *Journal of Statistical Software* 52:1–24.
- Bates, D., M. Mächler, B. Bolker, and S. Walker. 2015. Fitting linear mixed-effects models using lme4. *Journal of Statistical Software* 67:1–48.
- Bates, D., and M. Maechler. 2023. MatrixModels: Modelling with sparse and dense matrices.
- Bates, D., K. M. Mullen, J. C. Nash, and R. Varadhan. 2023. minqa: Derivative-free optimization algorithms by quadratic approximation.
- Beleites, C. 2020. arrayhelpers: Convenience functions for arrays.

References

- Bengtsson, H. 2023. `matrixStats`: Functions that apply to rows and columns of matrices (and to vectors).
- Bengtsson, H. 2024a. `globals`: Identify global objects in r expressions.
- Bengtsson, H. 2024b. `listenv`: Environments behaving (almost) as lists.
- Bengtsson, H. 2024c. `parallelly`: Enhancing the “parallel” package.
- Berwin A. Turlach R port by Andreas Weingessel <Andreas.Weingessel@ci.tuwien.ac.at> Fortran contributions from Cleve Moler `dpodi/LINPACK`), S. original by. 2019. `quadprog`: Functions to solve quadratic programming problems.
- Bolker, B. M., M. E. Brooks, C. J. Clark, S. W. Geange, J. R. Poulsen, M. H. H. Stevens, and J.-S. S. White. 2009. Generalized linear mixed models: A practical guide for ecology and evolution. *Trends in Ecology and Evolution* 24:127–135.
- Bolker, B., and D. Robinson. 2024. `broom.mixed`: Tidying methods for mixed models.
- Boshnakov, G. N. 2023. `Rdpack`: Update and manipulate rd documentation objects.
- Boshnakov, G. N., and C. Putman. 2023. `rbibutils`: Read “Bibtex” files and convert between bibliography formats.
- Bryan, J. 2016. `cellranger`: Translate spreadsheet cell ranges to rows and columns.
- Bryan, J., C. Citro, and H. Wickham. 2023. `gargle`: Utilities for working with google APIs.
- Bürkner, P.-C. 2017. `brms`: An R package for Bayesian multilevel models using Stan. *Journal of Statistical Software* 80:1–28.
- Bürkner, P.-C. 2018. Advanced Bayesian multilevel modeling with the R package `brms`. *The R Journal* 10:395–411.
- Bürkner, P.-C. 2021. Bayesian item response modeling in R with `brms` and Stan. *Journal of Statistical Software* 100:1–54.
- Bürkner, P.-C., J. Gabry, M. Kay, and A. Vehtari. 2023. `posterior`: Tools for working with posterior distributions.
- Chang, W. 2021. R6: Encapsulated classes with reference semantics.
- Chang, W. 2023a. `cachem`: Cache r objects with automatic pruning.
- Chang, W. 2023b. `fastmap`: Fast data structures.
- Chang, W., and J. Cheng. 2023. `later`: Utilities for scheduling functions to execute later with event loops.
- Chang, W., J. Cheng, J. Allaire, C. Sievert, B. Schloerke, Y. Xie, J. Allen, J. McPherson, A. Dipert, and B. Borges. 2024. `shiny`: Web application framework for r.
- Cheng, J. 2023. `promises`: Abstractions for promise-based asynchronous programming.
- Cheng, J., W. Chang, S. Reid, J. Brown, B. Trower, and A. Peslyak. 2024a. `httpuv`: HTTP and WebSocket server library.
- Cheng, J., T. Mastny, R. Iannone, B. Schloerke, and C. Sievert. 2024b. `sass`: Syntactically awesome style sheets (“Sass”).
- Cheng, J., and C. Sievert. 2023. `crosstalk`: Inter-widget interactivity for HTML widgets.
- Cheng, J., C. Sievert, B. Schloerke, W. Chang, Y. Xie, and J. Allen. 2024c. `htmltools`: Tools for HTML.
- Constantin, A.-E., and I. Patil. 2021. `ggsignif`: R package for displaying significance brackets for “ggplot2”. PsyArxiv.
- Corporation, M., and S. Weston. 2022. `doParallel`: Foreach parallel adaptor for the “parallel” package.
- Csardi, G. 2023a. `prettyunits`: Pretty, human readable formatting of quantities.
- Csardi, G. 2023b. `rematch`: Match regular expressions with a nicer “API”.
- Csardi, G., and S. Sorhus. 2015. `praise`: Praise users.
- Csárdi, G. 2019. `pkgconfig`: Private configuration for “R” packages.
- Csárdi, G. 2020. `rematch2`: Tidy output from regular expression matching.
- Csárdi, G. 2022. `crayon`: Colored terminal output.
- Csárdi, G., and W. Chang. 2024a. `callr`: Call r from r.
- Csárdi, G., and W. Chang. 2024b. `processx`: Execute and control system processes.
- Csárdi, G., and R. FitzJohn. 2023. `progress`: Terminal progress bars.
- Csárdi, G., J. Hester, H. Wickham, W. Chang, M. Morgan, and D. Tenenbaum. 2024. `remotes`: R package installation from remote repositories, including “GitHub”.
- Csárdi, G., K. Müller, and J. Hester. 2023. `desc`: Manipulate DESCRIPTION files.
- Dahl, D. B., D. Scott, C. Roosen, A. Magnusson, and J. Swinton. 2019. `xtable`: Export tables to LaTeX or HTML.

- Eddelbuettel, D. 2013. Seamless R and C++ integration with Rcpp. Springer, New York.
- Eddelbuettel, D., and J. J. Balamuta. 2018. Extending R with C++: A Brief Introduction to Rcpp. *The American Statistician* 72:28–36.
- Eddelbuettel, D., J. W. Emerson, and M. J. Kane. 2024a. BH: Boost c++ header files.
- Eddelbuettel, D., R. Francois, J. Allaire, K. Ushey, Q. Kou, N. Russell, I. Ucar, D. Bates, and J. Chambers. 2024b. Rcpp: Seamless r and c++ integration.
- Eddelbuettel, D., and R. François. 2011. Rcpp: Seamless R and C++ integration. *Journal of Statistical Software* 40:1–18.
- Elston, D. A., R. Moss, T. Boulinier, C. Arrowsmith, and X. Lambin. 2001. Analysis of aggregation, a worked example: Numbers of ticks on red grouse chicks. *Parasitology* 122:563–569.
- Fasiolo, M., S. N. Wood, M. Zaffran, R. Nedellec, and Y. Goude. 2021. qgam: Bayesian nonparametric quantile regression modeling in R. *Journal of Statistical Software* 100:1–31.
- FitzJohn, R. 2017. ids: Generate random identifiers.
- Fortran code by Alan Miller, T. L. based on. 2020. leaps: Regression subset selection.
- Fox, J., and S. Weisberg. 2019. An R companion to applied regression. Third. Sage, Thousand Oaks CA.
- Fox, J., S. Weisberg, and B. Price. 2022. carData: Companion to applied regression data sets.
- Francisco Rodriguez-Sanchez, and Connor P. Jackson. 2023. grateful: Facilitate citation of r packages.
- Gabry, J., B. Goodrich, M. Lysy, and A. Johnson. 2024. rstantools: Tools for developing r packages interfacing with “Stan”.
- Gabry, J., and T. Mahr. 2024. bayesplot: Plotting for bayesian models.
- Gabry, J., D. Simpson, A. Vehtari, M. Betancourt, and A. Gelman. 2019. Visualization in bayesian workflow. *J. R. Stat. Soc. A* 182:389–402.
- Gagolewski, M. 2022. stringi: Fast and portable character string processing in R. *Journal of Statistical Software* 103:1–59.
- Galili, T. 2015. dendextend: An r package for visualizing, adjusting, and comparing trees of hierarchical clustering. *Bioinformatics*.
- Garbett, S. P., J. Stephens, K. Simonov, Y. Xie, Z. Dong, H. Wickham, J. Horner, reikoch, W. Beasley, B. O’Connor, G. R. Warnes, M. Quinn, and Z. N. Kamvar. 2023. yaml: Methods to convert r data to YAML and back.
- Garnier, Simon, Ross, Noam, Rudis, Robert, Camargo, A. Pedro, Sciaini, Marco, Scherer, and Cédric. 2023. viridis(Lite) - colorblind-friendly color maps for r.
- Garnier, Simon, Ross, Noam, Rudis, Robert, Camargo, A. Pedro, Sciaini, Marco, Scherer, and Cédric. 2024. viridis(Lite) - colorblind-friendly color maps for r.
- Gaslam, B. 2021. diffobj: Diffs for r objects.
- Gaslam, B. 2023. fansi: ANSI control sequence aware string functions.
- Genz, A., and F. Bretz. 2009. Computation of multivariate normal and t probabilities. Springer-Verlag, Heidelberg.
- Gilbert, P., and R. Varadhan. 2019. numDeriv: Accurate numerical derivatives.
- Graves, S., H.-P. Piepho, and L. S. with help from Sundar Dorai-Raj. 2024. multcompView: Visualizations of paired comparisons.
- Gronau, Q. F., H. Singmann, and E.-J. Wagenmakers. 2020. bridgesampling: An R package for estimating normalizing constants. *Journal of Statistical Software* 92:1–29.
- Grosjean, P. 2024. SciViews-r. UMONS, MONS, Belgium.
- Hadfield, J. D. 2010. MCMC methods for multi-response generalized linear mixed models: The MCMCglmm R package. *Journal of Statistical Software* 33:1–22.
- Hadfield, J. D., A. J. Wilson, D. Garant, B. C. Sheldon, and L. E. Kruuk. 2010. The Misuse of BLUP in Ecology and Evolution. *American Naturalist* 175:116–125.
- Halekoh, U., and S. Højsgaard. 2014. A kenward-roger approximation and parametric bootstrap methods for tests in linear mixed models – the R package pbkrtest. *Journal of Statistical Software* 59:1–30.
- Hankin, R. K. S. 2007. Very large numbers in r: Introducing package brobdingnag. *R News* 7.
- Hartig, F. 2022. DHARMA: Residual diagnostics for hierarchical (multi-level / mixed) regression models.
- Hasselman, B. 2023. nleqslv: Solve systems of nonlinear equations.

References

- Henry, L., and H. Wickham. 2023. lifecycle: Manage the life cycle of your package functions.
- Henry, L., and H. Wickham. 2024. tidyselect: Select from a set of strings.
- Hester, J., and J. Bryan. 2024. glue: Interpreted string literals.
- Hester, J., and G. Csárdi. 2023. brio: Basic r input output.
- Hester, J., L. Henry, K. Müller, K. Ushey, H. Wickham, and W. Chang. 2024. withr: Run code “With” temporarily modified global state.
- Hester, J., H. Wickham, and J. Bryan. 2023a. vroom: Read and write rectangular text data quickly.
- Hester, J., H. Wickham, and G. Csárdi. 2023b. fs: Cross-platform file system operations based on “libuv”.
- Horst, A. M., A. P. Hill, and K. B. Gorman. 2020. palmerpenguins: Palmer archipelago (antarctica) penguin data.
- Houslay, T. M., and A. J. Wilson. 2017. Avoiding the misuse of BLUP in behavioural ecology. *Behavioral Ecology* 28:948–952.
- Hvitfeldt, E. 2022. emoji: Data and function to work with emojis.
- Iannone, R. 2023. fontawesome: Easily work with “Font Awesome” icons.
- Johnson, A. R. 2024. QuickJSR: Interface for the “QuickJS” lightweight “JavaScript” engine.
- Johnson, S. G. ? The NLOpt nonlinear-optimization package. ? ?
- Justin Talbot. 2023. labeling: Axis labeling.
- Kassambara, A. 2023a. ggpubr: “ggplot2” based publication ready plots.
- Kassambara, A. 2023b. rstatix: Pipe-friendly framework for basic statistical tests.
- Kassambara, A., and F. Mundt. 2020. factoextra: Extract and visualize the results of multivariate data analyses.
- Kay, M. 2023. tidybayes: Tidy data and geoms for Bayesian models.
- Kay, M. 2024a. ggdist: Visualizations of distributions and uncertainty in the grammar of graphics. *IEEE Transactions on Visualization and Computer Graphics* 30:414–424.
- Kay, M. 2024b. ggdist: Visualizations of distributions and uncertainty.
- Koenker, R. 2021. SparseM: Sparse linear algebra.
- Koenker, R. 2023. quantreg: Quantile regression.
- Kuznetsova, A., P. B. Brockhoff, and R. H. B. Christensen. 2017. lmerTest package: Tests in linear mixed effects models. *Journal of Statistical Software* 82:1–26.
- Lang, M. 2017. checkmate: Fast argument checks for defensive R programming. *The R Journal* 9:437–445.
- Lang, M., and R Core Team. 2021. backports: Reimplementations of functions introduced since r-3.0.0.
- Langfelder, P., and S. Horvath. 2012. Fast R functions for robust correlations and hierarchical clustering. *Journal of Statistical Software* 46:1–17.
- Lê, S., J. Josse, and F. Husson. 2008. FactoMineR: A package for multivariate analysis. *Journal of Statistical Software* 25:1–18.
- Lenth, R. 2024a. estimability: Tools for assessing estimability of linear predictions.
- Lenth, R. V. 2024b. emmeans: Estimated marginal means, aka least-squares means.
- Ligges, U., and M. Mächler. 2003. Scatterplot3d - an r package for visualizing multivariate data. *Journal of Statistical Software* 8:1–20.
- Lincoln, M. 2022. clipr: Read and write from the system clipboard.
- Loden, J., D. Daeschler, G. Rodola, and G. Csárdi. 2024. ps: List, query, manipulate system processes.
- Lüdecke, D., M. S. Ben-Shachar, I. Patil, P. Waggoner, and D. Makowski. 2021. performance: An R package for assessment, comparison and testing of statistical models. *Journal of Open Source Software* 6:3139.
- Lüdecke, D., P. Waggoner, and D. Makowski. 2019. insight: A unified interface to access information from model objects in R. *Journal of Open Source Software* 4:1412.
- Makowski, D., M. S. Ben-Shachar, and D. Lüdecke. 2019. bayestestR: Describing effects and their uncertainty, existence and significance within the bayesian framework. *Journal of Open Source Software* 4:1541.
- Microsoft, and S. Weston. 2022. foreach: Provides foreach looping construct.
- Müller, K. 2023. rprojroot: Finding files in project subdirectories.
- Murdoch, D., and E. D. Chow. 2023. ellipse: Functions for drawing ellipses and ellipse-like confidence regions.
- Narasimhan, B., S. G. Johnson, T. Hahn, A. Bouvier, and K. Kiêu. 2023. cubature: Adaptive multivariate integration over hypercubes.

- Neuwirth, E. 2022. RColorBrewer: ColorBrewer palettes.
- O’Hara-Wild, M., M. Kay, and A. Hayes. 2024. distributional: Vectorised probability distributions.
- Oehlschlägel, J., and B. Ripley. 2022. bit: Classes and methods for fast memory-efficient boolean selections.
- Oehlschlägel, J., and L. Silvestri. 2020. bit64: A S3 class for vectors of 64bit integers.
- Ooms, J. 2023a. askpass: Password entry utilities for r, git, and SSH.
- Ooms, J. 2023b. openssl: Toolkit for encryption, signatures and certificates based on OpenSSL.
- Ooms, J. 2023c. sys: Powerful and reliable tools for running system commands in r.
- Ooms, J. 2024a. commonmark: High performance CommonMark and github markdown rendering in r.
- Ooms, J. 2024b. curl: A modern and flexible web client for r.
- Paradis, E., and K. Schliep. 2019. Ape 5.0: An environment for modern phylogenetics and evolutionary analyses in R. *Bioinformatics* 35:526–528.
- Patil, I., D. Makowski, M. S. Ben-Shachar, B. M. Wiernik, E. Bacher, and D. Lüdecke. 2022. datawizard: An R package for easy data preparation and statistical transformations. *Journal of Open Source Software* 7:4684.
- Pedersen, T. L. 2023. textshaping: Bindings to the “HarfBuzz” and “Fribidi” libraries for text shaping.
- Pedersen, T. L. 2024. patchwork: The composer of plots.
- Pedersen, T. L., B. Nicolae, and R. François. 2022. farver: High performance colour space manipulation.
- Pedersen, T. L., J. Ooms, and D. Govett. 2024. systemfonts: System native font finding.
- Perry, P. O. 2023. utf8: Unicode text processing.
- Plate, T., and R. Heiberger. 2016. abind: Combine multidimensional arrays.
- Plummer, M., N. Best, K. Cowles, and K. Vines. 2006. CODA: Convergence diagnosis and output analysis for MCMC. *R News* 6:7–11.
- Potter, S. 2012. Introducing the selectr package. The University of Auckland, Auckland, New Zealand.
- R Core Team. 2024. R: A language and environment for statistical computing. R Foundation for Statistical Computing, Vienna, Austria.
- R Special Interest Group on Databases (R-SIG-DB), H. Wickham, and K. Müller. 2024. DBI: R database interface.
- Ratnakumar, S., T. Mick, and T. Davis. 2021. rappdirs: Application directories: Determine where to save data, caches, and logs.
- Sarkar, D. 2008. *Lattice: Multivariate data visualization with r*. Springer, New York.
- Schafer, J., R. Opgen-Rhein, V. Zuber, M. Ahdesmaki, A. P. D. Silva, and K. Strimmer. 2021. corpcor: Efficient estimation of covariance and (partial) correlation.
- Sievert, C. 2020. Interactive web-based data visualization with r, plotly, and shiny. Chapman; Hall/CRC.
- Sievert, C., and J. Cheng. 2021. jquerylib: Obtain “jQuery” as an HTML dependency object.
- Sievert, C., J. Cheng, and G. Aden-Buie. 2024. bslib: Custom “Bootstrap” “Sass” themes for “shiny” and “rmarkdown”.
- Sklyar, O., D. Murdoch, M. Smith, D. Eddebuettel, R. Francois, K. Soetaert, and J. Ranke. 2021. inline: Functions to inline c, c++, fortran function calls from r.
- Slowikowski, K. 2024. ggrepel: Automatically position non-overlapping text labels with “ggplot2”.
- Solymos, P., and Z. Zawadzki. 2023. pbapply: Adding progress bar to “*apply” functions.
- Spinu, V. 2024. timechange: Efficient manipulation of date-times.
- Stan Development Team. 2020. StanHeaders: Headers for the R interface to Stan.
- Stan Development Team. 2021. Stan modeling language users guide and reference manual, 2.26.
- Stan Development Team. 2024. RStan: The R interface to Stan.
- Stauffer, R., G. J. Mayr, M. Dabernig, and A. Zeileis. 2009. Somewhere over the rainbow: How to make effective use of colors in meteorological visualizations. *Bulletin of the American Meteorological Society* 96:203–216.
- Stoffel, M. A., S. Nakagawa, and H. Schielzeth. 2017. rptR: Repeatability estimation and variance decomposition by generalized linear mixed-effects models. *Methods in Ecology and Evolution* 8:1639–1644.
- The VSNi Team. 2023. asreml: Fits linear mixed models using REML.
- Urbanek, S. 2015. base64enc: Tools for base64 encoding.
- Urbanek, S., and T. Ts’o. 2024. uuid: Tools for generating and handling of UUIDs.
- Ushey, K. 2023. sourcetools: Tools for reading, tokenizing and parsing r code.


References

- Ushey, K., and H. Wickham. 2024. `renv`: Project environments.
- Vaidyanathan, R., Y. Xie, J. Allaire, J. Cheng, C. Sievert, and K. Russell. 2023. `htmlwidgets`: HTML widgets for R.
- van den Boogaart, K. G. 2023. `tensorA`: Advanced tensor arithmetic with named indices.
- Vaughan, D. 2023. `tzdb`: Time zone database information.
- Vaughan, D., and M. Dancho. 2022. `furrr`: Apply mapping functions in parallel using futures.
- Vaughan, D., J. Hester, and R. François. 2023. `cpp11`: A C++11 interface for R's C interface.
- Vehtari, A., J. Gabry, M. Magnusson, Y. Yao, P.-C. Bürkner, T. Paananen, and A. Gelman. 2024. `loo`: Efficient leave-one-out cross-validation and WAIC for Bayesian models.
- Vehtari, A., A. Gelman, and J. Gabry. 2017. Practical Bayesian model evaluation using leave-one-out cross-validation and WAIC. *Statistics and Computing* 27:1413–1432.
- Vehtari, A., A. Gelman, D. Simpson, B. Carpenter, and P.-C. Bürkner. 2021. Rank-normalization, folding, and localization: An improved rhat for assessing convergence of MCMC (with discussion). *Bayesian Analysis*.
- Venables, B., K. Hornik, and M. Maechler. 2022. `polynom`: A collection of functions to implement a class for univariate polynomial manipulations.
- Venables, W. N., and B. D. Ripley. 2002. *Modern applied statistics with S*. Fourth. Springer, New York.
- Wei, T., and V. Simko. 2021. R package “`corrplot`”: Visualization of a correlation matrix.
- Wickham, C. 2018. `munsell`: Utilities for using munsell colours.
- Wickham, H. 2007. Reshaping data with the `reshape` package. *Journal of Statistical Software* 21:1–20.
- Wickham, H. 2011a. The split-apply-combine strategy for data analysis. *Journal of Statistical Software* 40:1–29.
- Wickham, H. 2011b. `testthat`: Get started with testing. *The R Journal* 3:5–10.
- Wickham, H. 2019. `lazyeval`: Lazy (non-standard) evaluation.
- Wickham, H. 2021. `ellipsis`: Tools for working with ...
- Wickham, H. 2023a. `blob`: A simple S3 class for representing vectors of binary data (“BLOBS”).
- Wickham, H. 2023b. `waldo`: Find differences between R objects.
- Wickham, H., M. Averick, J. Bryan, W. Chang, L. D. McGowan, R. François, G. Golemund, A. Hayes, L. Henry, J. Hester, M. Kuhn, T. L. Pedersen, E. Miller, S. M. Bache, K. Müller, J. Ooms, D. Robinson, D. P. Seidel, V. Spinu, K. Takahashi, D. Vaughan, C. Wilke, K. Woo, and H. Yutani. 2019. Welcome to the tidyverse. *Journal of Open Source Software* 4:1686.
- Wickham, H., W. Chang, J. Hester, and L. Henry. 2024a. `pkgload`: Simulate package installation and attach.
- Wickham, H., and G. Golemund. 2016. *R for Data Science: Import, Tidy, Transform, Visualize, and Model Data*. 1st edition. O’Reilly Media.
- Wickham, H., L. Henry, and D. Vaughan. 2023a. `vctrs`: Vector helpers.
- Wickham, H., J. Hester, W. Chang, K. Müller, and D. Cook. 2021. `memoise`: “Memoisation” of functions.
- Wickham, H., J. Hester, and G. Csárdi. 2024b. `pkgbuild`: Find tools needed to build R packages.
- Wickham, H., M. Kuhn, and D. Vaughan. 2022a. `generics`: Common S3 generics not provided by base R methods related to model fitting.
- Wickham, H., and T. L. Pedersen. 2023. `gtable`: Arrange “Grobs” in tables.
- Wickham, H., T. L. Pedersen, and D. Seidel. 2023b. `scales`: Scale functions for visualization.
- Wickham, H., C. O. Wilke, and T. L. Pedersen. 2022b. `isoband`: Generate isolines and isobands from regularly spaced elevation grids.
- Wickham, H., and Y. Xie. 2023. `evaluate`: Parsing and evaluation tools that provide more details than the default.
- Wilke, C. O. 2024a. `cowplot`: Streamlined plot theme and plot annotations for “`ggplot2`”.
- Wilke, C. O. 2024b. `ggribes`: Ridgeline plots in “`ggplot2`”.
- Wolak, M. E. 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.
- Xiao, N. 2024. `ggsci`: Scientific journal and sci-fi themed color palettes for “`ggplot2`”.
- Xie, Y. 2014. `knitr`: A comprehensive tool for reproducible research in R. *in* V. Stodden, F. Leisch, and R. D. Peng, editors. *Implementing reproducible computational research*. Chapman; Hall/CRC.
- Xie, Y. 2015. *Dynamic documents with R and knitr*. 2nd edition. Chapman; Hall/CRC, Boca Raton, Florida.
- Xie, Y. 2016. `bookdown`: Authoring books and technical documents with R markdown. Chapman; Hall/CRC, Boca

- Raton, Florida.
- Xie, Y. 2019. TinyTeX: A lightweight, cross-platform, and easy-to-maintain LaTeX distribution based on TeX live. TUGboat 40:30–32.
- Xie, Y. 2021. mime: Map filenames to MIME types.
- Xie, Y. 2023. knitr: A general-purpose package for dynamic report generation in r.
- Xie, Y. 2024a. bookdown: Authoring books and technical documents with r markdown.
- Xie, Y. 2024b. tinytex: Helper functions to install and maintain TeX live, and compile LaTeX documents.
- Xie, Y. 2024c. xfun: Supporting functions for packages maintained by “Yihui Xie”.
- Xie, Y., J. J. Allaire, and G. Golemund. 2018. R markdown: The definitive guide. Chapman; Hall/CRC, Boca Raton, Florida.
- Xie, Y., J. Cheng, and X. Tan. 2024. DT: A wrapper of the JavaScript library “DataTables”.
- Xie, Y., C. Dervieux, and E. Riederer. 2020. R markdown cookbook. Chapman; Hall/CRC, Boca Raton, Florida.
- Xie, Y., and Y. Qiu. 2022. highr: Syntax highlighting for r source code.
- Yao, Y., A. Vehtari, D. Simpson, and A. Gelman. 2017. Using stacking to average bayesian predictive distributions. Bayesian Analysis.
- Zeileis, A., J. C. Fisher, K. Hornik, R. Ihaka, C. D. McWhite, P. Murrell, R. Stauffer, and C. O. Wilke. 2020. colorspace: A toolbox for manipulating and assessing colors and palettes. Journal of Statistical Software 96:1–49.
- Zeileis, A., and G. Grothendieck. 2005. zoo: S3 infrastructure for regular and irregular time series. Journal of Statistical Software 14:1–27.
- Zeileis, A., K. Hornik, and P. Murrell. 2009. Escaping RGBland: Selecting colors for statistical graphics. Computational Statistics & Data Analysis 53:3259–3270.
- Zeileis, A., and T. Hothorn. 2002. Diagnostic checking in regression relationships. R News 2:7–10.
- Zhao, J. H. 2007. gap: Genetic analysis package. Journal of Statistical Software 23:1–18.
- Zhao, J. H. 2023a. gap: Genetic analysis package.
- Zhao, J. H. 2023b. gap.datasets: Datasets for “gap”.

A. R

Need to write something about R

We also use various methods for manipulating and visualising data frames using the  tidyverse (Wickham et al. 2019) (including `tidyr`, `dplyr`, `ggplot2` etc). You can get more details on their use can be found at in the Book R for Data Science (Wickham and Grolemund 2016) which is freely available as a bookdown website [here](#).

