

BIO8940 Advanced stats and Open Science

Work in Progress

Julien Martin

26-02-2021

Table des Matières

Note	5
Préface	7
Quelques points importants à retenir	7
Qu'est-ce que R et pourquoi l'utiliser dans ce cours?	8
Installation des logiciels nécessaires	8
Instructions générales pour les laboratoires	10
Notes sur le manuel	10
 I Open Science	 13
1 Introduction to open Science	15
2 Introduction to Rmarkdown	17
2.1 Practical	17
3 Introduction to github with R	21
3.1 Practical	21
 II Statistics	 25
4 Refresher on glm	27
5 Introduction to linear mixed models	29
5.1 Practical	29
6 Introduction to GLMM	33
6.1 Practical	33
6.2 Conclusions	54

III	Appendix	57
R		59
To do list		61

Note



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.

Préface

Les exercices de laboratoire que vous retrouverez dans les pages qui suivent sont conçus de manière à vous permettre de développer une expérience pratique en analyse de données à l'aide d'un logiciel (R). R est un logiciel très puissant, mais comme tous les logiciels, il a des limites. En particulier il ne peut réfléchir à votre place, vous dire si l'analyse que vous tentez d'effectuer est appropriée ou sensée, ou interpréter biologiquement les résultats.

Quelques points importants à retenir

- Avant de commencer une analyse statistique, il faut d'abord vous familiariser son fonctionnement. Cela ne veut pas dire que vous devez connaître les outils mathématiques qui la sous-tendent, mais vous devriez au moins comprendre les principes utilisés lors de cette analyse. Avant de faire un exercice de laboratoire, lisez donc la section correspondante dans les notes de cours. Sans cette lecture préalable, il est très probable que les résultats produits par le logiciel, même si l'analyse a été effectuée correctement, seront indéchiffrables.
- Les laboratoires sont conçus pour compléter les cours théoriques et vice versa. À cause des contraintes d'horaires, il se pourrait que le cours et le laboratoire ne soient pas parfaitement synchronisés. N'hésitez donc pas à poser des questions sur le labo en classe ou des questions théoriques au laboratoire.
- Travaillez sur les exercices de laboratoire à votre propre rythme. Certains exercices prennent beaucoup moins de temps que d'autres et il n'est pas nécessaire de compléter un exercice par séance de laboratoire. En fait deux séances de laboratoire sont prévues pour certains des exercices. Même si vous n'êtes pas notés sur les exercices de laboratoire, soyez conscient que ces exercices sont essentiels. Si vous ne les faites pas, il est très peu probable que vous serez capable de compléter les devoirs et le projet de session. Prenez donc ces exercices de laboratoire au sérieux !
- Les 2 premier laboratoires sont conçu pour vous permettre d'acquérir ou de réviser le minimum de connaissances requises pour vous permettre de réaliser les exercices de laboratoires avec R. Il y a presque toujours de multiples façons de faire les choses avec R et vous ne trouverez ici que des méthodes simples. Ceux et celles d'entre vous qui y sont enclins pourront trouver en ligne des instructions plus détaillées et complexes. En particulier, je vous conseille :
 - R pour les débutants http://cran.r-project.org/doc/contrib/Paradis-rdebuts_fr.pdf

- An introduction to R <http://cran.r-project.org/doc/manuals/R-intro.html>
- Si vous préférez des manuels, le site web de CRAN en garde une liste commentée à : <http://www.r-project.org/doc/bib/R-books.html>
- Une liste impressionnante de très bon livre sur R <https://www.bigbookofr.com/>
- Finalement, comme aide-mémoire à garder sous la main, je vous recommande R reference card par Tom Short <http://cran.r-project.org/doc/contrib/Short-refcard.pdf>

Qu'est-ce que R et pourquoi l'utiliser dans ce cours?

R est un logiciel libre et multi-plateforme formant un système statistique et graphique. R est également un langage de programmation spécialisé pour les statistiques.

R a deux très grands avantages pour ce cours, et un inconvénient embêtant initialement mais qui vous forcera à acquérir des excellentes habitudes de travail. Le premier avantage est que vous pouvez tous l'installer sur votre (ou vos) ordinateurs personnel gratuitement. C'est important parce que c'est à l'usage que vous apprendrez et maîtriserez réellement les biostatistiques et cela implique que vous devez avoir un accès facile et illimité à un logiciel statistique. Le deuxième avantage est que R peut tout faire en statistiques. R est conçu pour être extensible et est devenu l'outil de prédilection des statisticiens mondialement. La question n'est plus : " Est-ce que R peut faire ceci? ", mais devient " Comment faire ceci avec R ". Et la recherche internet est votre ami. Aucun autre logiciel n'offre ces deux avantages.

L'inconvénient embêtant initialement est que l'on doit opérer R en tapant des instructions (ou en copiant des sections de code) plutôt qu'en utilisant des menus et en cliquant sur différentes options. Si on ne sait pas quelle commande taper, rien ne se passe. Ce n'est donc pas facile d'utilisation à priori. Cependant, il est possible d'apprendre rapidement à faire certaines des opérations de base (ouvrir un fichier de données, faire un graphique pour examiner ces données, effectuer un test statistique simple). Et une fois que l'on comprend le principe de la chose, on peut assez facilement trouver sur le web des exemples d'analyses ou de graphiques plus complexes et adapter le code à nos propres besoins. C'est ce que vous ferez dans le premier laboratoire pour vous familiariser avec R.

Pourquoi cet inconvénient est-il d'une certaine façon un avantage? Parce que vous allez sauver du temps en fin de compte. Garanti. Croyez-moi, on ne fait jamais une analyse une seule fois. En cours de route, on découvre des erreurs d'entrée de données, ou que l'on doit faire l'analyse séparément pour des sous-groupes, ou on obtient des données supplémentaires, ou on fait une erreur. On doit alors recommencer l'analyse. Avec une interface graphique et des menus, cela implique recommencer à cliquer ici, entre des paramètres dans des boîtes et sélectionner des boutons. Chaque fois avec possibilité d'erreur. Avec une série de commandes écrites, il suffit de corriger ce qui doit l'être puis de copier-coller l'ensemble pour répéter instantanément. Et vous avez la possibilité de parfaitement documenter ce que vous avez fait. C'est comme cela que les professionnels travaillent et offrent une assurance de qualité de leurs résultats.

Installation des logiciels nécessaires

R

Pour installer R sur un nouvel ordinateur, allez au site <http://cran.r-project.org/>. Vous y trouverez des versions compilées (binaries) ou non (sources) pour votre système d'exploitation de prédilection (Windows, MacOS, Linux).

Note : R a déjà été installé sur les ordinateurs du laboratoire (la version pourrait être un peu plus ancienne, mais cela devrait être sans conséquences).

0.0.1 Text editor or IDE

Tinn-r Atom sublime, emacs, vim

Rstudio

RStudio est un environnement de développement intégré (IDE) créé spécifiquement pour travailler avec R. Sa popularité connaît une progression foudroyante depuis 2014. Il permet de consulter dans une interface conviviale ses fichiers de script, la ligne de commande R, les rubriques d'aide, les graphiques, etc.

RStudio est disponible à l'identique pour les plateformes Windows, OS X et Linux. Pour une utilisation locale sur son poste de travail, on installera la version libre (Open Source) de RStudio Desktop depuis le site <https://www.rstudio.com/products/rstudio/download/>

Visual Studio Code

Tinn-r

Paquets pour R

- Rmarkdown
- tinytex

Ces 2 paquets devraient être installés automatiquement avec RStudio, mais pas toujours. Je vous recommande donc de les installer manuellement. Pour ce faire, simplement copier-coller le texte suivant dans le terminal R.

```
install.packages(c("rmarkdown", "tinytex"))
```

pandoc

laTeX

- tinytex or others

Instructions générales pour les laboratoires

- Apporter une clé USB ou son équivalent à chaque séance de laboratoire pour sauvegarder votre travail.
- Lire l'exercice de laboratoire AVANT la séance, lire le code R correspondant et préparer vos questions sur le code.
- Durant les pré-labs, écouter les instructions et posez vos questions au moment approprié.
- Faites les exercices du manuel de laboratoire à votre rythme, en équipe, puis je vous recommande de commencer (compléter?) le devoir. Profitez de la présence du démonstrateur et du prof...
- Pendant vos analyses, copiez-collez des fragments de sorties de R dans un document (par exemple dans votre traitement de texte favori) et annotez abondamment.
- Ne tapez pas directement vos commandes dans R mais plutôt dans un script. Vous pourrez ainsi refaire le labo instantanément, récupérer des fragments de code, ou plus facilement identifier les erreurs dans vos analyses.
- Créez votre propre librairie de fragments de codes (snippets). Annotez-là abondamment. Vous vous en félicitez plus tard.

Notes sur le manuel

Vous trouverez dans le manuel des explications sur la théorie, du code R, des explications sur R et des exercices.

Le manuel essaie aussi de mettre en évidence le texte de différentes manières.



Avec des sections à vous de jouer, ui indique un exercice à faire, idéalement sans regarder la solution qui se trouve plus bas.



des avertissements



des avertissements



des points importants



des notes



et des conseils

Resources {-}

Ce document est généré par l'excellente extension [bookdown](#) de [Yihui Xie](#). Il est basé sur le précédent manuel de laboratoire *BIO4558 manuel de laboratoire* par Antoine Morin. L'introduction à R est largement reprise de l'excellent manuel de **Julien Barnier** intitulé [Introduction à R et au tidyverse](#)

Licence

Ce document est mis à disposition selon les termes de la [Licence Creative Commons Attribution - Pas d'Utilisation Commerciale - Partage dans les Mêmes Conditions 4.0 International](#).



Figure 1: Licence Creative Commons

Partie I

Open Science

Chapitre 1

Introduction to open Science

Chapitre 2

Introduction to Rmarkdown

2.1 Practical

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

2.1.1 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_~ body_mass_g sex
##   <fct>   <fct>         <dbl>         <dbl>         <int>         <int> <fct>
## 1 Adelie  Torge~           39.1           18.7           181           3750 male
## 2 Adelie  Torge~           39.5           17.4           186           3800 fema~
## 3 Adelie  Torge~           40.3            18           195           3250 fema~
## 4 Adelie  Torge~           NA            NA            NA            NA <NA>
## 5 Adelie  Torge~           36.7           19.3           193           3450 fema~
## 6 Adelie  Torge~           39.3           20.6           190           3650 male
## # ... with 1 more variable: year <int>
```

2.1.2 Questions

1) Install the package `palmerpenguins`.

```
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.

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

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

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

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 `length()`. Calculate the mean of the (numeric) traits measured on the penguins.

```
## Numerical exploration
```

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

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.

```
## 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)
```
```

```
## 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)
`r`
```

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 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 pulldown 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](#)

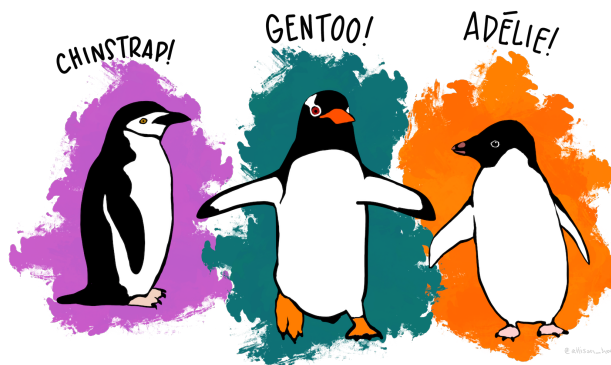


Figure 2.1: Happy coding

Chapitre 3

Introduction to github with R

3.1 Practical

3.1.1 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 .

3.1.2 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_~ body_mass_g sex
##   <fct>   <fct>         <dbl>         <dbl>         <int>      <int> <fct>
## 1 Adelie  Torge~           39.1           18.7           181        3750 male
## 2 Adelie  Torge~           39.5           17.4           186        3800 fema~
## 3 Adelie  Torge~           40.3            18           195        3250 fema~
## 4 Adelie  Torge~            NA            NA            NA            NA <NA>
```

```
## 5 Adelie Torge~          36.7          19.3          193          3450 fema~
## 6 Adelie Torge~          39.3          20.6          190          3650 male
## # ... with 1 more variable: 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~ Anvers Torge~ Adul~ N1A1
## 2 PAL0708          2 Adelie~ Anvers Torge~ Adul~ N1A2
## 3 PAL0708          3 Adelie~ Anvers Torge~ Adul~ N2A1
## 4 PAL0708          4 Adelie~ Anvers Torge~ Adul~ N2A2
## 5 PAL0708          5 Adelie~ Anvers Torge~ Adul~ N3A1
## 6 PAL0708          6 Adelie~ Anvers Torge~ Adul~ N3A2
## # ... with 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.

3.1.3 Questions

- 1) Create a github account if not done yet.
- 2) Configure Rstudio with your github account using the `usethis` package.
- 3) Store your GITHUB Personal Authorisation Token in your `.Renv` file
- 4) Create a new R Markdown project, 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 accep 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 3.1: Happy git(hub)-ing

Partie II

Statistics

Chapitre 4

Refresher on glm

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

Exercice 5

On a relevé les notes en maths, anglais et sport d'une classe de 6 élèves et on a stocké ces données dans trois vecteurs :

```
maths <- c(12, 16, 8, 18, 6, 10)
anglais <- c(14, 9, 13, 15, 17, 11)
sport <- c(18, 11, 14, 10, 8, 12)
```

Calculer la moyenne des élèves de la classe en anglais.

Calculer la moyenne générale de chaque élève.

```
## [1] 13.16667
```

```
## [1] 14.66667 12.00000 11.66667 14.33333 10.33333 11.00000
```

Exercice 5

On a relevé les notes en maths, anglais et sport d'une classe de 6 élèves et on a stocké ces données dans trois vecteurs :

```
maths <- c(12, 16, 8, 18, 6, 10)
anglais <- c(14, 9, 13, 15, 17, 11)
sport <- c(18, 11, 14, 10, 8, 12)
```

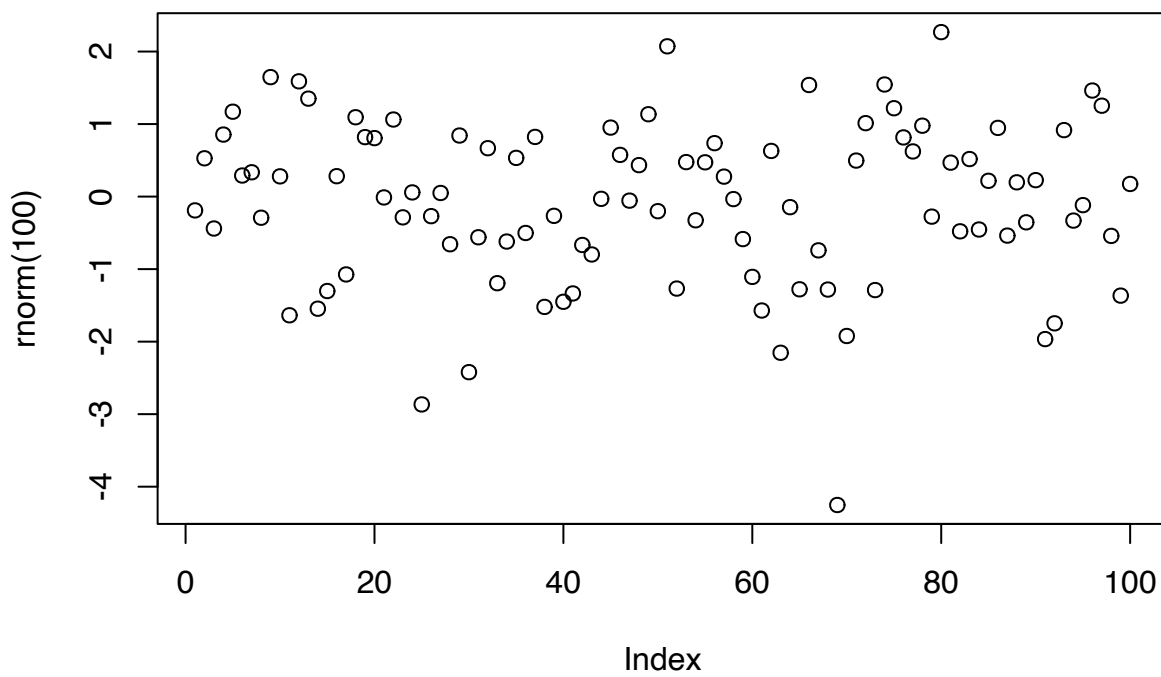
Calculer la moyenne des élèves de la classe en anglais.

Calculer la moyenne générale de chaque élève.

Test pour montrer la solution

```
## [1] 13.16667
```

```
## [1] 14.66667 12.00000 11.66667 14.33333 10.33333 11.00000
```



Chapitre 5

Introduction to linear mixed models

5.1 Practical

5.1.1 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 Munro baggers enjoying their whisky after a long day in the hills.

5.1.1.1 Experimental design - what is the structure of the data we have?

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 male experienced the three instruder 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).

5.1.1.2 Data

Let’s load the data file and make sure we understand what it contains

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

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

- 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)

5.1.2 Questions

- 1) Load the libraries (`lme4`)
- 2) Load the data `unicorns.csv` and look at the summary and data structure
- 3) Fit a first mixed model with `lmer` that have only individual identity as a random effect and only a population mean.
- 4) Look at the output of the model (`summary()`)
- 5) This is a fairly rubbish model so now fit a better model by adding opponent size (`opp_size`) and block (`block`) as fixed effects. Look at the output of the model (`summary()`)
- 6) Where are the **p-values** ? what I have done wrong. Load `lmerTest` and refit the model using the exact same code. Look at the summary again.
- 7) Testing for random effects using `ranova()`. You can also do it by hand if you want.
- 8) Estimate repeatability, either by hand or using the `rpt` package
- 9) Now what about the effect on an individual body size on its aggression. Do a new model including body size also as a fixed effects. Look at the model output, estimate the probability associated with the random effects.
- 10) Make a few diagnostic plots before you can get too excited by your results (*homoscedasticity, Gaussian distribution for residuals, linear relation, similar within group variance*)
- 11) Compare repeatability among different models.

5.1.3 Happy mixed-modelling



Figure 5.2: The superb unicorn

Chapitre 6

Introduction to GLMM

6.1 Practical

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).

6.1.1 Packages and functions

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

```
library(lme4)
library(plyr)
library(reshape)
library(tidyverse)
library(gridExtra)
library(lattice)
source("data/glmm_funs.R")
```

6.1.2 The data set

In this data set, the response variable is the number of fruits (i.e. seedcapsules) 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. apicalmeristem 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.

6.1.3 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 `transform()`, 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
## 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
## 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
## 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
## 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
## 5.SP  0  0  0  0  0  0  0  0  0 43 22 12  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  0 13 24 14  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 45 47 45
## 8.SP  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0 13 16 35  0  0  0  0  0  0
```

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. For example, `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 - try the commands `colSums(reptab>0)` and `table(colSums(reptab>0))` (and the equivalent for rowSums) and figure out what they are telling you. - Using this recipe, how would you compute the range of number of genotypes per treatment combination?

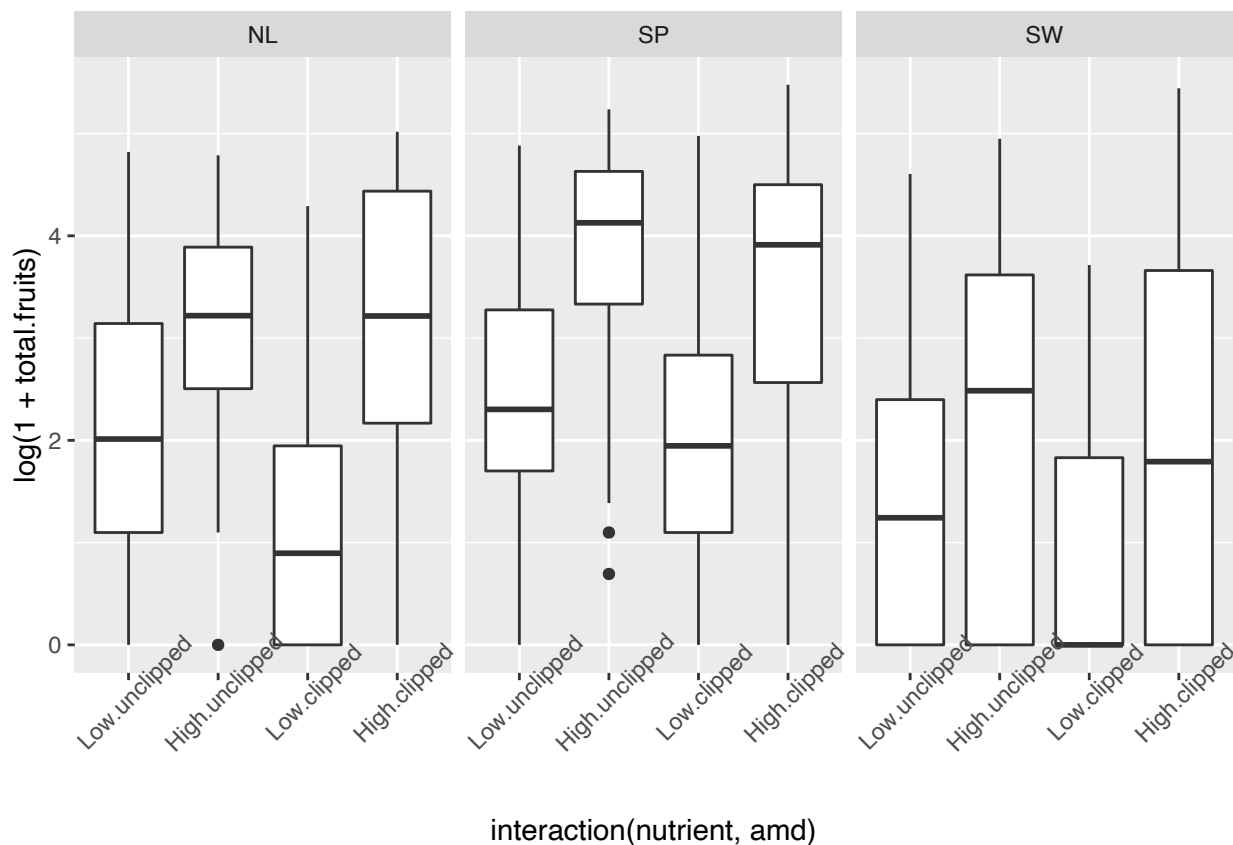
This reveals that we have only 2–4 populations per region and 2–3 genotypes per population. However, we also have 2–13 replicates per genotype for each treatment combination (four unique treatment combinations: 2 levels of nutrients by 2 levels of simulated herbivory). Thus, even though this was a reasonably 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.

We have only two random effects (population, individual), and so Laplace or Gauss-Hermite Quadrature (GHQ) should suffice, rather than requiring more complex methods. 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.

6.1.4 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).

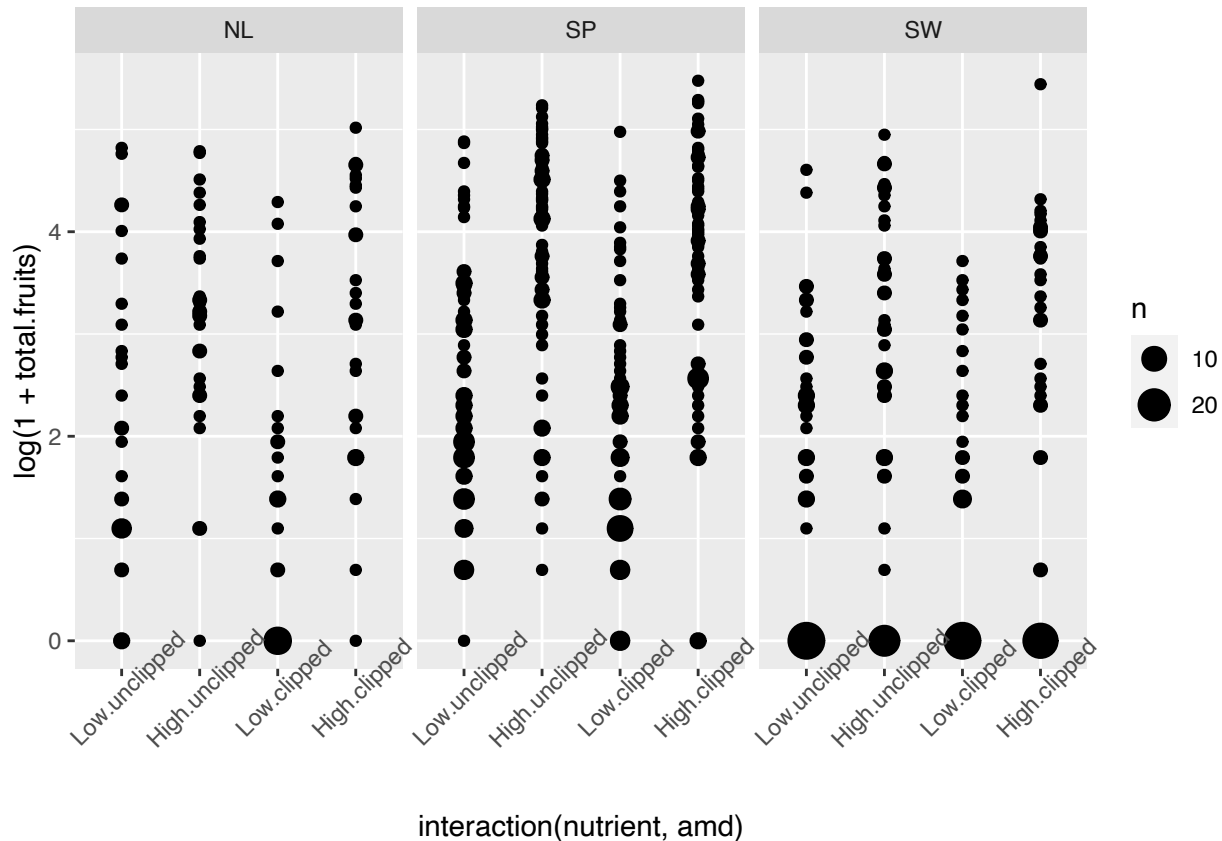
```
qplot(
  interaction(nutrient, amd),
  log(1 + total.fruits),
  data = dat_tf, geom = "boxplot") +
  facet_wrap(~reg, nrow = 1) +
  theme(axis.text.x = element_text(angle = 45))
```



```

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))

```



Exercise generate these plots and figure out how they work before continuing. Try conditioning/faceting on population rather than region: for ggplot 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.

6.1.5 Choose an error distribution

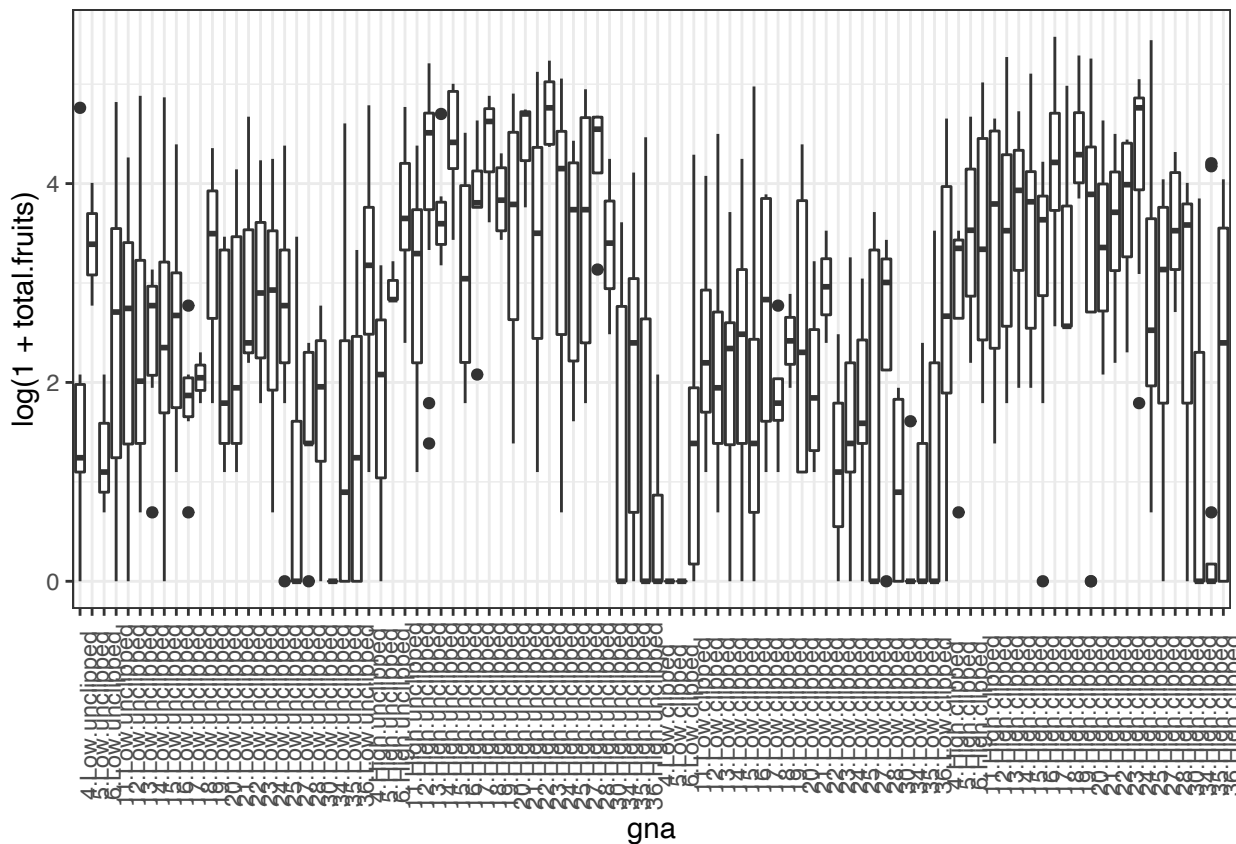
The data are non-normal in principle (i.e., count data, so our first guess would be a Poisson distribution). If we transform total fruits with the canonical link function (log), we hope to see relatively homogeneous variances across categories and groups.

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 = interaction(gen, nutrient, amd),
    gna = reorder(gna, 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))
```



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 (mean) for each genotype \times treatment combination and provide a statistical summary of each group's .

```
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 $\text{Var} = \mu$ (a line through the origin) with $\mu > 1$ is called a quasi-Poisson pattern (this term describes the mean-variance relationship, not any particular probability distribution); we can implement it statistically via quasi-likelihood (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, $\text{Var} = \mu(1 + \mu)$ 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`.

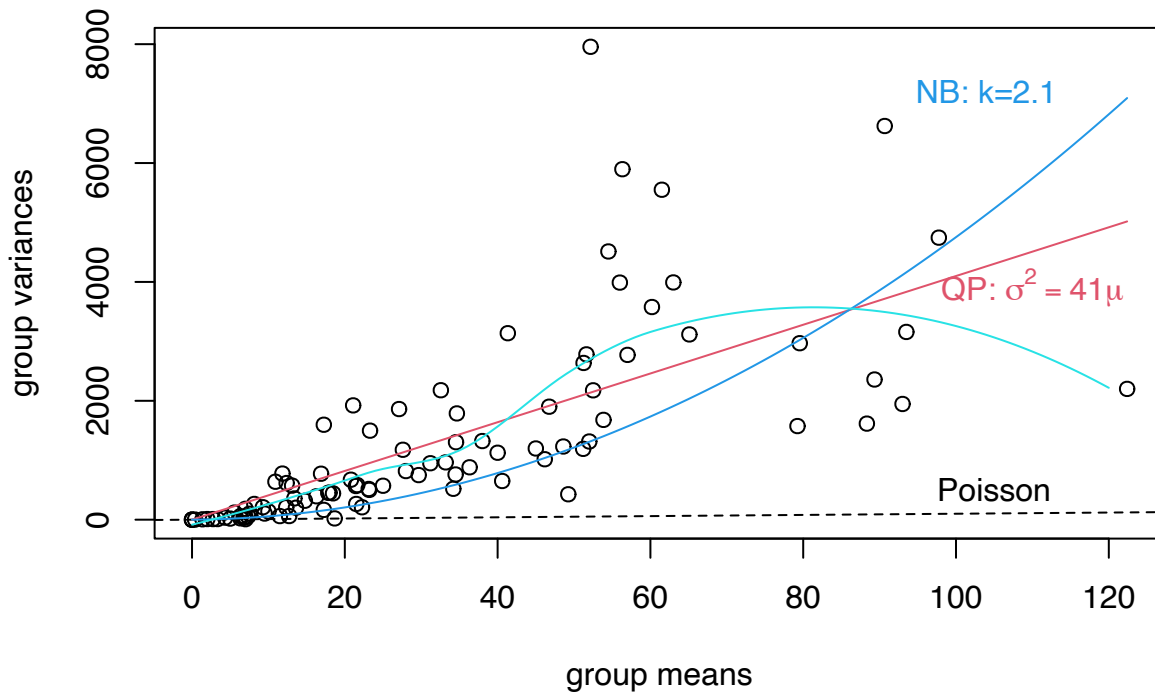
```
lm1 <- lm(grp_vars ~ grp_means - 1) ## `quasi-Poisson' fit
phi_fit <- coef(lm1)
lm2 <- lm((grp_vars - grp_means) ~ I(grp_means^2) - 1)
k_fit <- 1 / coef(lm2)
```

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)

```



Same with ggplot

```

ggplot(
  data.frame(grp_means, grp_vars),
  aes(x = grp_means, y = grp_vars)) +

```



```

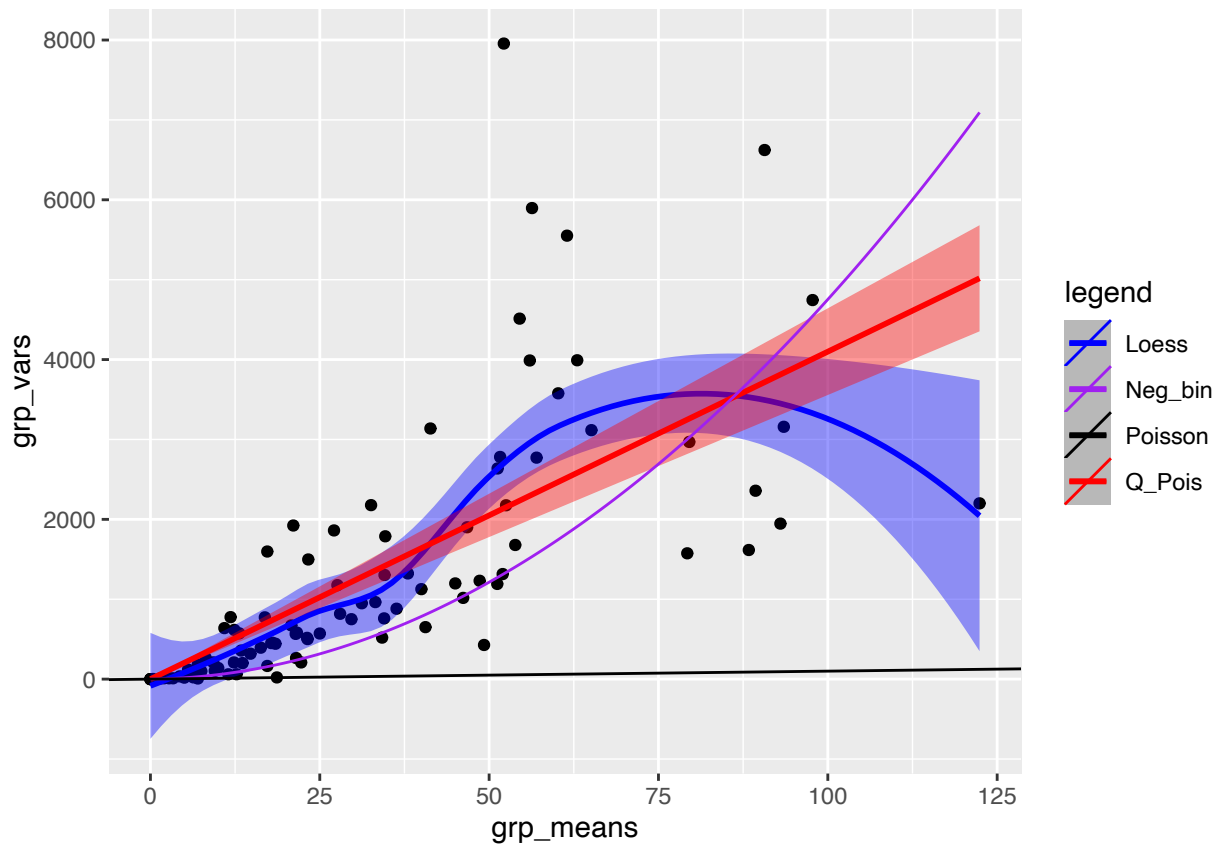
geom_point() +
geom_smooth(aes(colour = "Loess", fill = "Loess")) +
geom_smooth(
  method = "lm", formula = y ~ x - 1,
  aes(colour = "Q_Pois", fill = "Q_Pois")) +
stat_function(
  fun = function(x) x * (1 + x / k_fit),
  aes(colour = "Neg_bin", fill = "Neg_bin")
) +
geom_abline(
  aes(intercept = 0, slope = 1, colour = "Poisson", fill = "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: Ignoring unknown aesthetics: fill
```

```
## Warning: Ignoring unknown aesthetics: fill
```

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



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

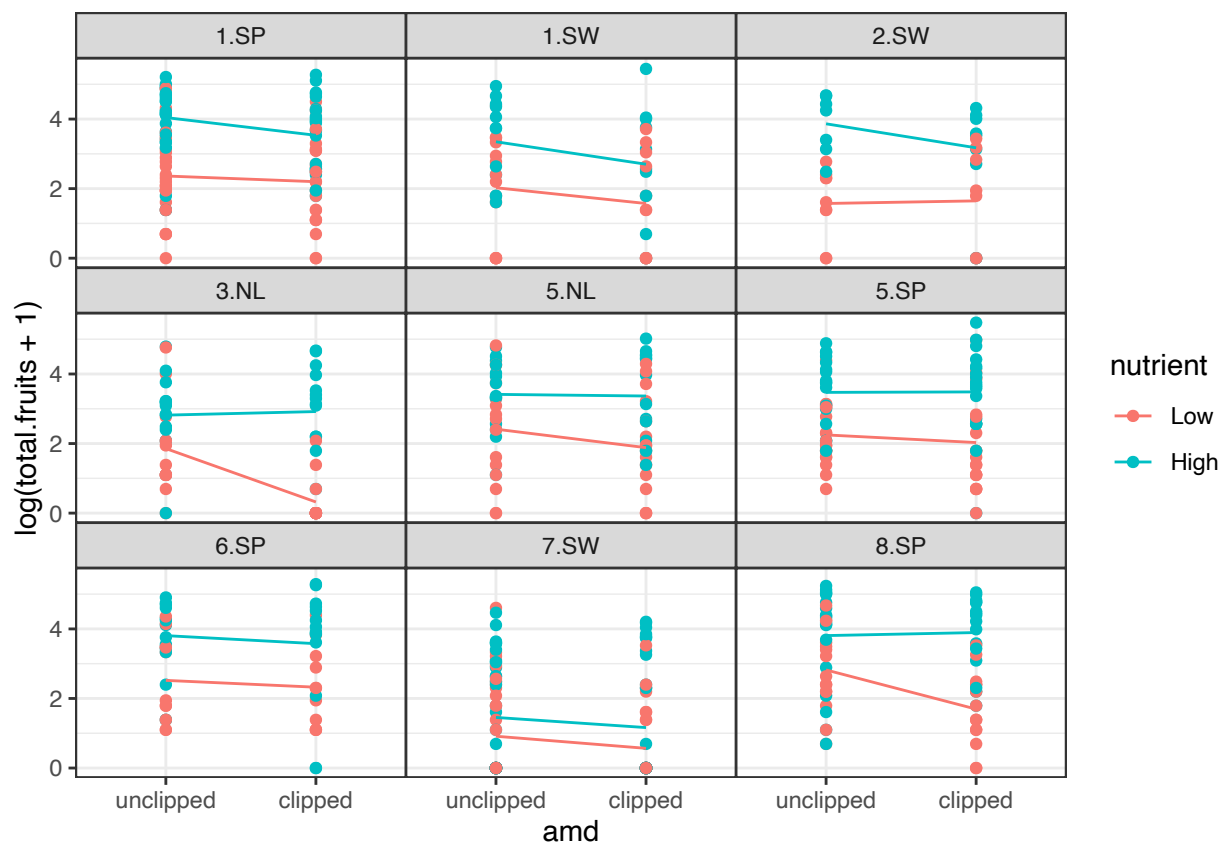
- compare a simple quadratic fit to the data (i.e., without the linear part) with the negative binomial and quasipoisson fits

6.1.5.1 Plotting the response vs treatments

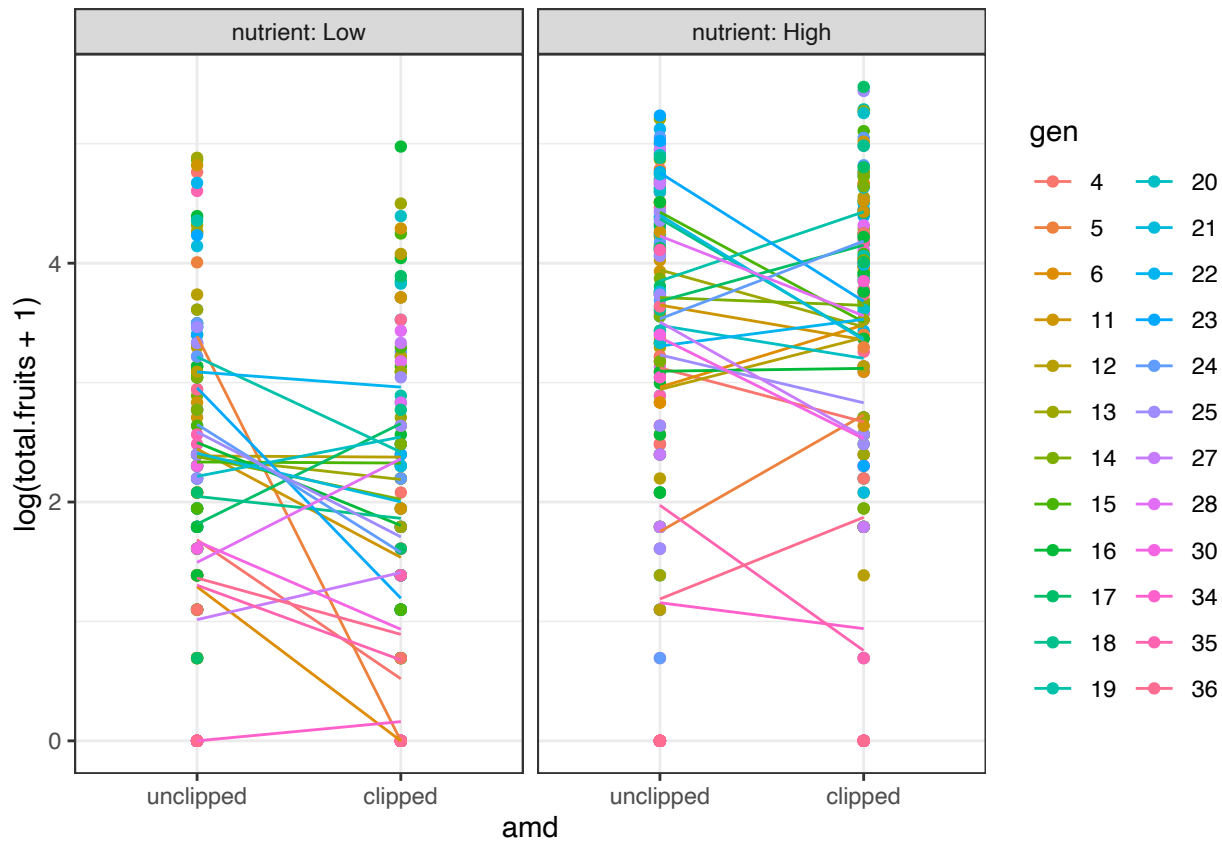
Just to avoid surprises

```
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() +
  ggplot2::theme(panel.margin = unit(0, "lines")) +
  facet_wrap(~popu)
```

```
## Warning: `panel.margin` is deprecated. Please use `panel.spacing` property
## instead
```



```
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)
```

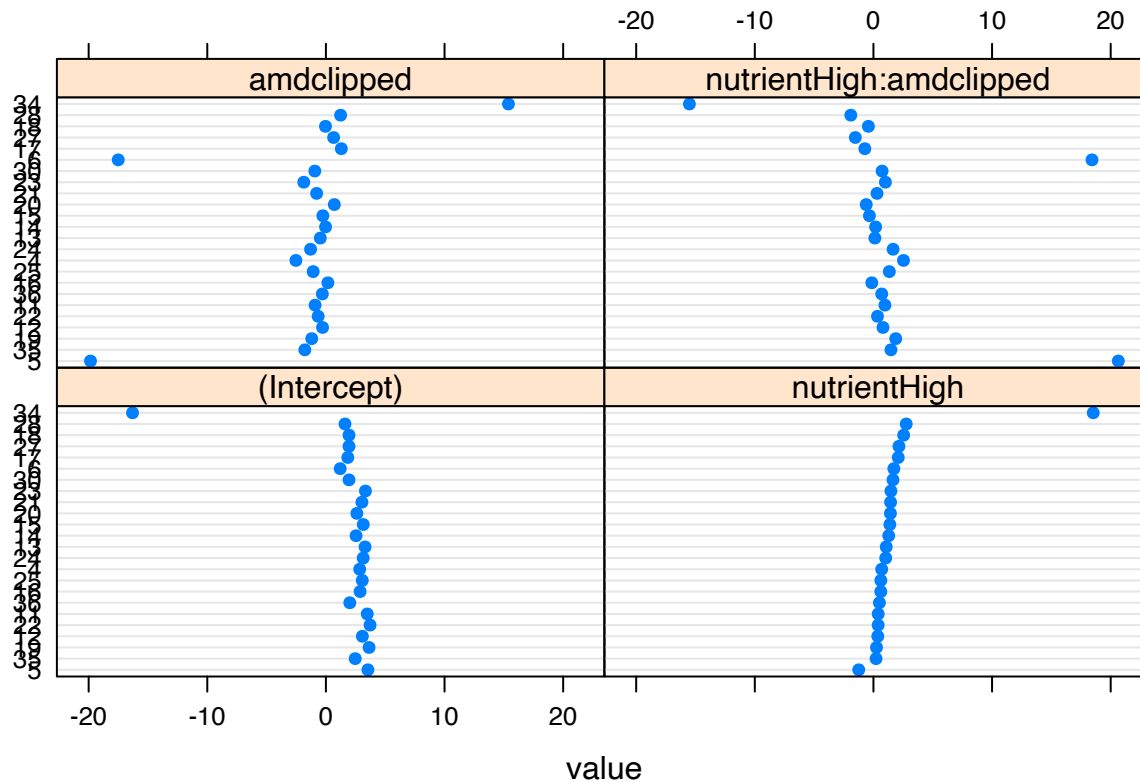


6.1.6 Fitting group-wise GLM

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)
```

```
## Using grp as id variables
```



Exercise Fit plot to examine output

6.1.7 Fitting and evaluating GLMMs

Now we (try to) build and fit a full model, using `glmer` in the `lme4` package⁹. 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| = 0.0132432 (tol = 0.002, component 1)
```

```
overdisp_fun(mp1)
```

```
##          chisq          ratio          p
## 13909.46562    23.25998    0.00000
```

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

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 (function (fn, par, lower = rep.int(-Inf, n), upper = rep.int(Inf, :
## failure to converge in 10000 evaluations
```

```
## Warning in optwrap(optimizer, devfun, start, rho$lower, control = control, :
## convergence code 4 from Nelder_Mead: failure to converge in 10000 evaluations
```

```
## Warning in checkConv(attr(opt, "derivs"), opt$par, ctrl = control$checkConv, :
## Model failed to converge with max|grad| = 0.169728 (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 -0.9991408  -0.9877061
## amdclipped          -0.9991408  1.0000000   0.9916060
## nutrientHigh        -0.9877061  0.9916060   1.0000000
## amdclipped:nutrientHigh 0.8225706 -0.8397183  -0.9012402
##          amdclipped:nutrientHigh
## (Intercept)          0.8225706
## amdclipped          -0.8397183
## nutrientHigh        -0.9012402
## 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| = 0.17819 (tol = 0.002, component 1)
```

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

```
##           (Intercept) amdclipped nutrientHigh
## (Intercept)  1.0000000 -0.7762341  -0.9979863
## amdclipped  -0.7762341  1.0000000   0.7743002
## nutrientHigh -0.9979863  0.7743002   1.0000000
```

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

```
##           (Intercept) amdclipped nutrientHigh
## (Intercept)  1.0000000  0.9995678   0.9973568
## amdclipped   0.9995678  1.0000000   0.9961872
## nutrientHigh  0.9973568  0.9961872   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 using `update`, fit the models with (1) clipping variation at both genotype and population levels; (2) nutrient variation at both genotype and populations; using `printvc`, convince yourself that trying to fit variation in either clipping or nutrients leads to overfitting (perfect correlations). 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

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

```
## Warning in checkConv(attr(opt, "derivs"), opt$par, ctrl = control$checkConv, :
## Model failed to converge with max|grad| = 0.0256392 (tol = 0.002, component 1)
```

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 observationlevel random effect should have taken care of it):

```
overdisp_fun(mp4)
```

```
##      chisq      ratio      p
## 177.3529154  0.2883787  1.0000000
```

6.1.8 Inference

6.1.8.1 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) 1.43206
## gen    (Intercept) 0.28728
## popu   (Intercept) 0.80614
```


6.1.8.1.1 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 21twice 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 |
## mp4v1:      popu) + nutrient:amd
## mp4: total.fruits ~ nutrient + amd + rack + status + (1 | X) + (1 |
## mp4:      gen) + (1 | popu) + nutrient:amd
##      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 4.0627  1    0.04384 *
## ---
## 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 |
## mp4v2:      gen) + nutrient:amd
## mp4: total.fruits ~ nutrient + amd + rack + status + (1 | X) + (1 |
## mp4:      gen) + (1 | popu) + nutrient:amd
##      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 18.211  1 1.977e-05 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

For various forms of linear mixed models, the RLLRsim 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)
## raply in plyr is a convenient wrapper for repeating the simulation many times
nulldist <- raply(200, simdev(), .progress = "text")
## 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)
```

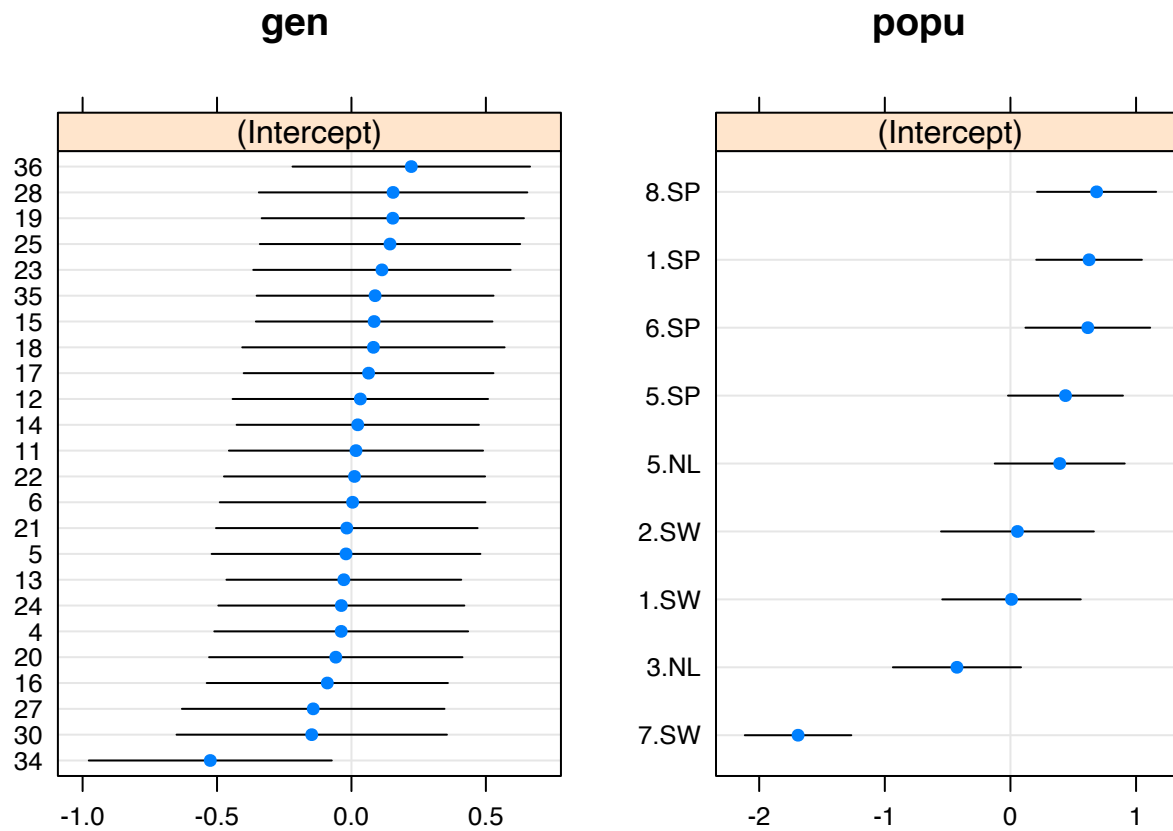
```
## [1] 0.01492537
```

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.

```
library(lattice)
pp <- list(
  layout.widths = list(left.padding = 0, right.padding = 0),
  layout.heights = list(top.padding = 0, bottom.padding = 0)
```

```
)
r1 <- ranef(mp4, condVar = TRUE)
d1 <- lattice::dotplot(r1, par.settings = pp)
print(grid.arrange(d1$gen, d1$popu, nrow = 1))
```



```
## TableGrob (1 x 2) "arrange": 2 grobs
##   z      cells   name                                grob
## 1 1 (1-1,1-1) arrange lattice[GRID.lattice.4255]
## 2 2 (1-1,2-2) arrange lattice[GRID.lattice.4256]
```

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.

6.1.8.2 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| = 0.00763683 (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 55.081
## status             2  1.611
## nutrient:amd       1  1.443
```

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

```
## Warning in checkConv(attr(opt, "derivs"), opt$par, ctrl = control$checkConv, :
## Model failed to converge with max|grad| = 0.00763683 (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 57.081 4.182e-14 ***
## status             2 5017.0  5.611  0.06047 .
## nutrient:amd       1 5016.8  3.443  0.06353 .
## ---
## 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| = 0.00763683 (tol = 0.002, component 1)
```

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

```
## Data: dat_tf  
## Models:  
## mp5: total.fruits ~ nutrient + amd + rack + status + (1 | X) + (1 |  
## mp5:      gen) + (1 | popu)  
## mp4: total.fruits ~ nutrient + amd + rack + status + (1 | X) + (1 |  
## mp4:      gen) + (1 | popu) + nutrient:amd  
##      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 3.4427  1    0.06353 .  
## ---  
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Exercise Test now the reduced model.

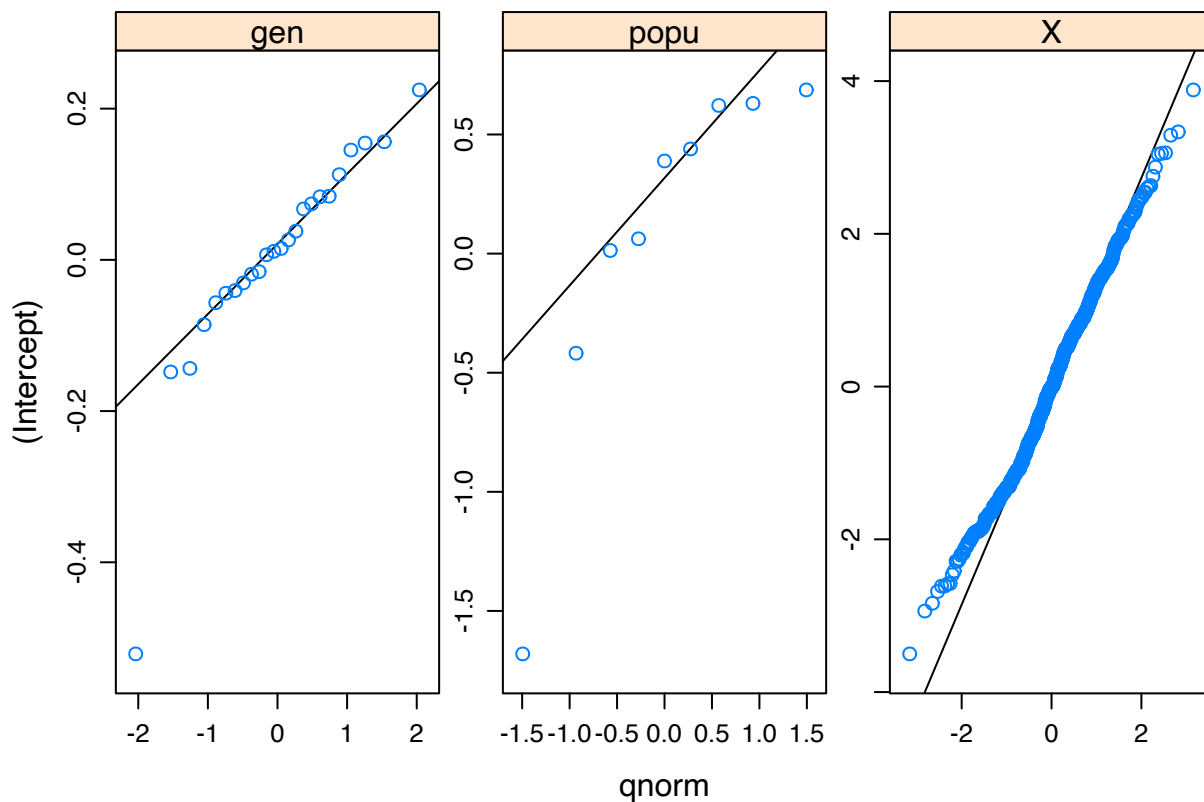
In the reduced model, we find that both nutrients and clipping have strong effects, whether measured by AIC or LRT. If we wanted to be still more careful about our interpretation, we would try to relax the asymptotic assumption. In classical linear models, we would do this by doing F tests with the appropriate 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). We use `ldply` from the `reshape` package to collapse the list of random effect values into a data frame (we might have to do something different if there were more than one random effect within each level, e.g. a model including (nutrient|gen)). The fancy panel code in the figure adds a reference line to the Q-Q plot.

```

re_stack <- plyr::ldply(ranef(mp5))
print(qqmath(~ `(Intercept)` | .id,
  data = re_stack, scales = list(relation = "free"),
  prepanel = prepanel.qqmathline,
  panel = function(x, ...) {
    panel.qqmathline(x, ...)
    panel.qqmath(x, ...)
  },
  layout = c(3, 1)
))

```



6.2 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

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

Partie III

Appendix

R

Need to write something about R

To do list

- clean code
- add more info about allEffects and plotting prediction of models
- convert all plots to ggplot
- convert data handling to tidyverse
- finish chapters