GLMMs LAB: gene-by-environment interaction in total fruit production of wild populations of *Arabidopsis thaliana*

Ben Bolker and Hank Stevens

August 18, 2014

1 Introduction

This lab is a variant of the (revised/in revision) supplement by Bolker *et al.* of 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 individuals to better tolerate herbivory. We use GLMMs to estimate the effect of nutrient levels on tolerance to herbivory in *Arabidopsis thaliana* (fixed effects), and the extent to which populations vary in their responses (variance components)[1].

In this example, we step deliberately through the process of model building and analysis. The primary approach uses `glmer` from the `lme4` package to analyze the data as a lognormal-Poisson hierarchical model, but we also use other approaches.

Load packages:

```r
library(lme4)
library(bbmle) ## for AICtab
library(coefplot2)
library(plyr)
```

[1]Data and context are provided courtesy of J. Banta and M. Pigliucci, State University of New York (Stony Brook).

```
library(reshape2)
library(gridExtra)
library(MCMCglmm)
library(ggplot2); theme_set(theme_bw())
source("../R/glmm_funs.R") ## utility functions
```

Later we will also load the `glmmADMB` package, but there are conflicts between some of its methods and `lme4`, so we will until we really need it.

2 The 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) is 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 genotype. The data also include 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.

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         : Factor w/ 3 levels "NL","SP","SW": 1 1 1 1 1 1 1 1 1 1 ...
##  $ popu        : Factor w/ 9 levels "1.SP","1.SW",..: 4 4 4 4 4 4 4 4 4 4 ...
##  $ 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         : Factor w/ 2 levels "clipped","unclipped": 1 1 1 1 1 2 1 1 2 1 ...
##  $ status      : Factor w/ 3 levels "Normal","Petri.Plate",..: 3 2 1 1 3 2 1 1 1 2
##  $ 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 coud either redo the `read.csv` command with `colClasses` set, or set the variables to factors by hand.

- 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 <- transform(dat.tf,
                    X=factor(X),
                    gen=factor(gen),
                    rack=factor(rack),
                    nutrient=factor(nutrient),
                    amd=factor(amd,levels=c("unclipped","clipped")))
```

2

The above output shows that the data include,

`X` observation number.p

`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

[2] There can be a tradeoff between clarity and comprehension. More laboriously but perhaps more clearly, we could have made the same changes to the integer-coded factors as follows:

```
dat.tf$X <- factor(dat.tf$X)
dat.tf$gen <- factor(dat.tf$gen)
dat.tf$rack <- factor(dat.tf$rack)
dat.tf$nutrient <- factor(dat.tf$nutrient)
```

I find the code with `transform` clearer, but you should use whatever form makes more sense to you (and is easier to understand when you come back to the code later).

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
##   1.SP  0  0  0  0  0 39 26 35  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
##   2.SW  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0 18 14  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
##   5.NL  0  0  0 35 26  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
##   6.SP  0  0  0  0  0  0  0  0  0  0  0 13 24 14  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
##   8.SP  0  0  0  0  0  0  0  0  0  0  0  0  0  0 13 16 35  0  0  0  0  0
##       gen
## popu   35 36
##   1.SP  0  0
##   1.SW  0  0
##   2.SW  0  0
##   3.NL  0  0
##   5.NL  0  0
##   5.SP  0  0
##   6.SP  0  0
##   7.SW 47 45
##   8.SP  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?

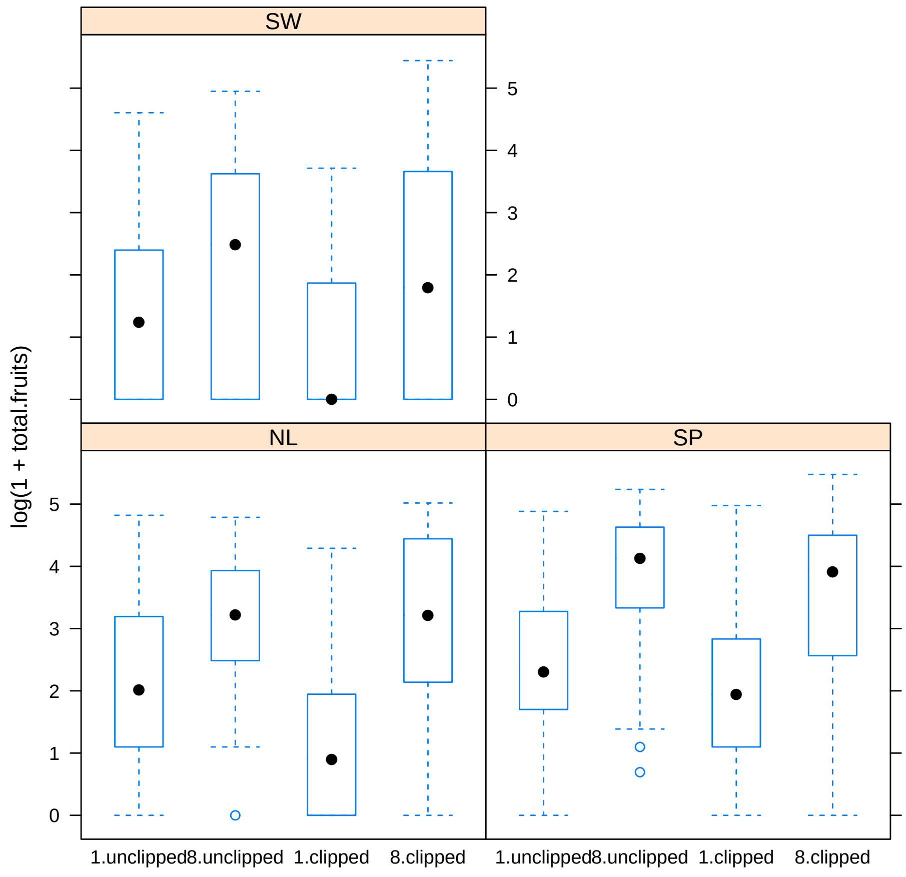
This reveals that we have only 2–4 populations per region and 2–3 genotypes per population. However, we also have 2–13 replicates per genotype per treatment combination (four unique treatment combinations: 2 levels of nutrients by 2 levels of simulated herbivory). Thus, even though this was a reasonably large experiment (625 plants), there were a very small number of replicates with which to estimate variance components at the region level: we are going to ignore region (although we could alternatively treat it as a fixed effect, while still including population as a random effect).

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, we will have to deal with overdispersion.

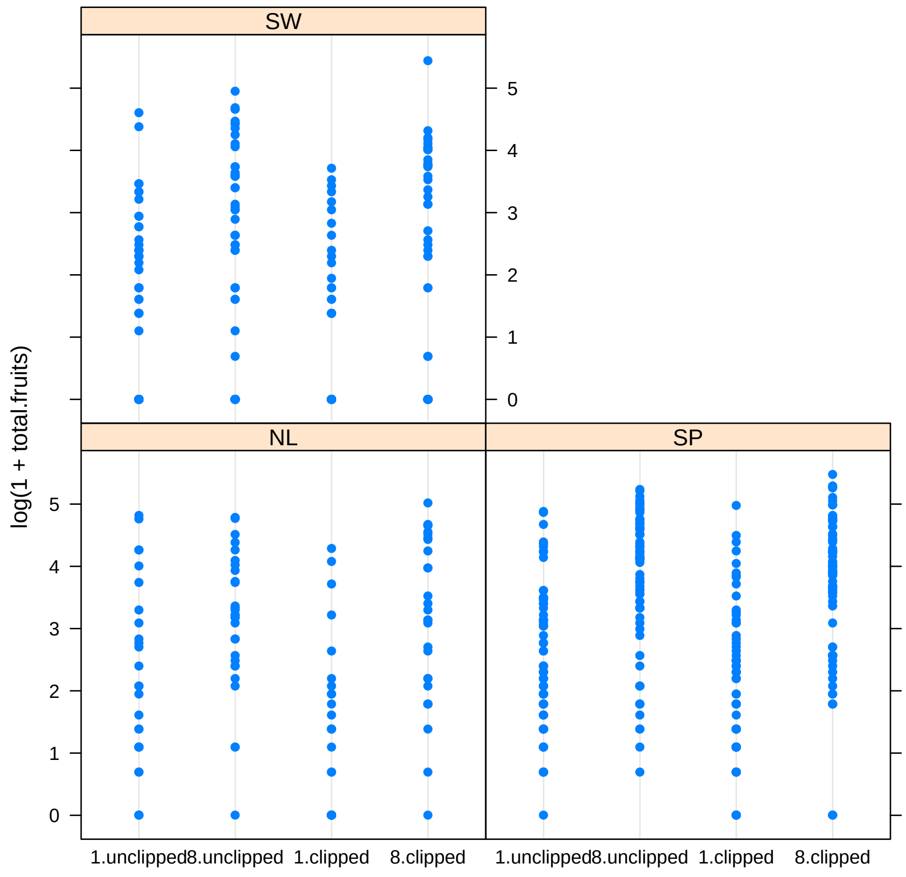
3 Look at overall patterns in the data

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):

```
bwplot(log(1+total.fruits)~interaction(nutrient,amd)|reg,data=dat.tf)
```

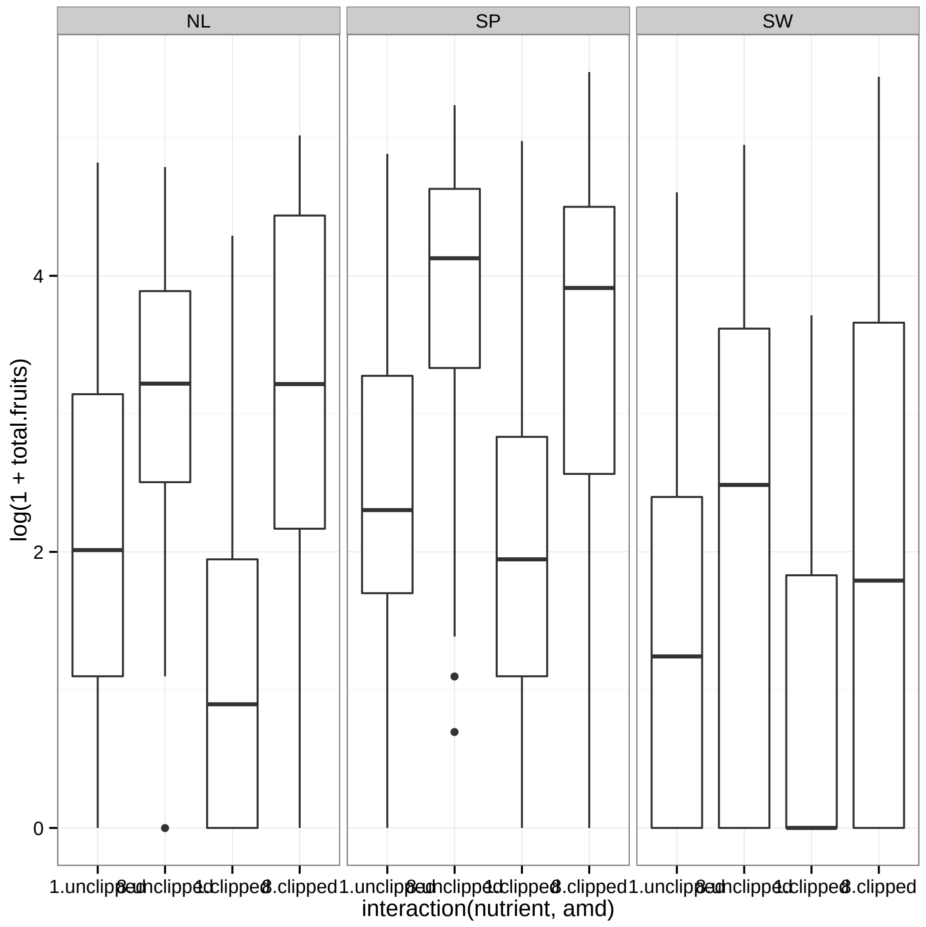


```
dotplot(log(1+total.fruits)~interaction(nutrient,amd)|reg,data=dat.tf)
```

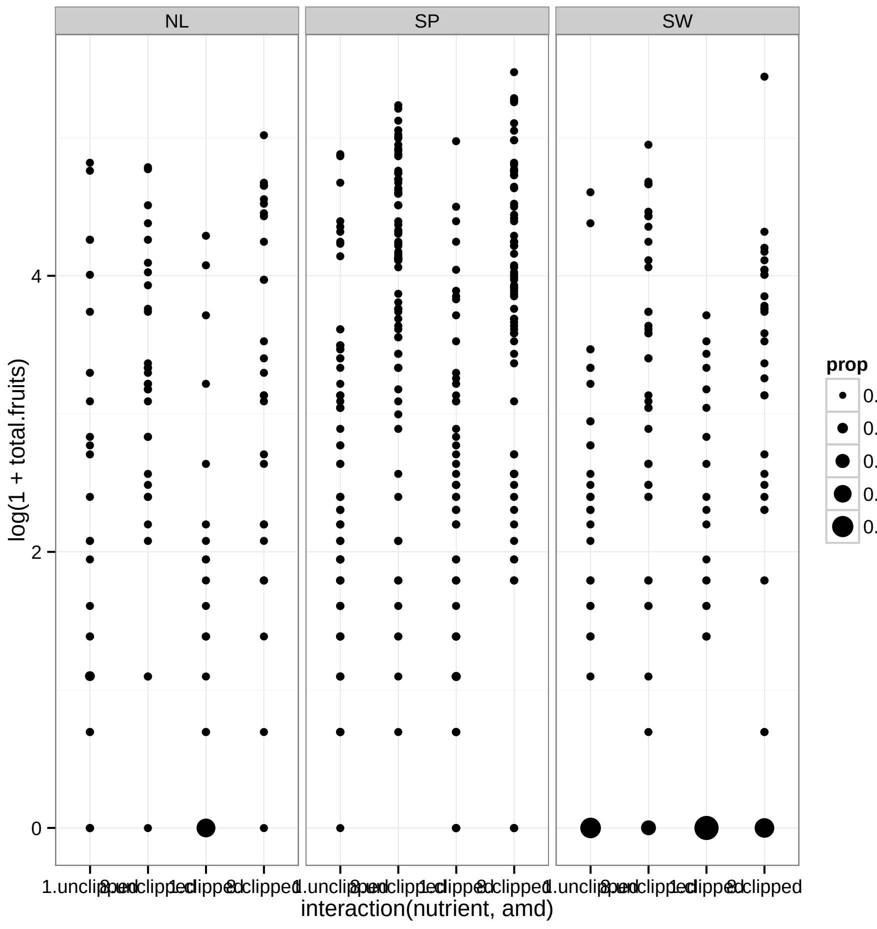


`ggplot` will do just about the same thing. Its `stat_sum` operator is useful for displaying duplicated points . . .

```
qplot(interaction(nutrient,amd),log(1+total.fruits),
      data=dat.tf,geom="boxplot")+facet_wrap(~reg,nrow=1)
```



```
qplot(interaction(nutrient,amd),log(1+total.fruits),
      data=dat.tf)+facet_wrap(~reg,nrow=1)+stat_sum()
```



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. For extra credit, reorder the subplots by overall mean fruit set and/or colour the points according to the region they come from . . .

4 Choose an error distribution; graphical checks

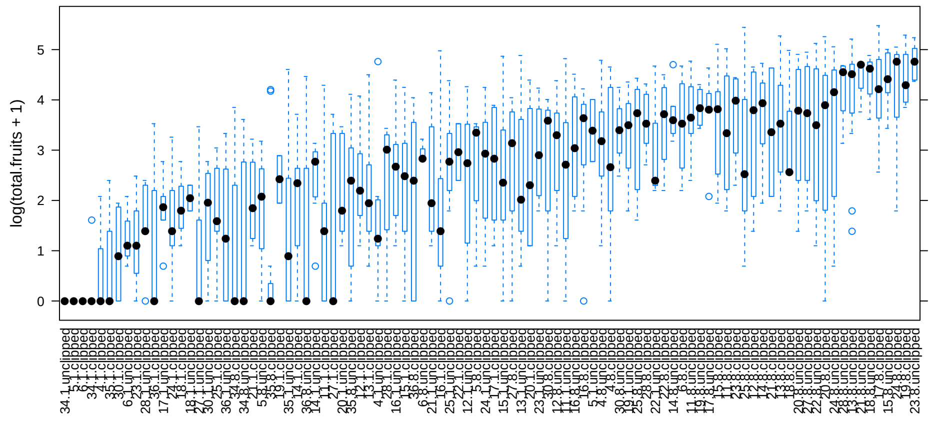
The data are non-normal in principle (i.e., count data, so our first guess would be a Poisson distribution). If we transform total fruits with the canonical link function (log), we hope to see relatively homogeneous variances across categories and groups.

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

```
## use within() to make multiple changes within a data frame
dat.tf <- within(dat.tf,
                 {
                   gna <- interaction(gen,nutrient,amd)
                   gna <- reorder(gna, total.fruits, mean)
                 })
```

Now we use the `bwplot` function in the `lattice` package to generate a box-and-whisker plot of log(fruits + 1).

```
bwplot(log(total.fruits + 1) ~  gna,
              data=dat.tf,
              scales=list(x=list(rot=90))  ## rotate x-axis labels
              )
```



3

[3]Alternatively, one can also use the **ggplot2** package to draw these graphs. In this case, we rotate the graph 90 degrees (using **coord_flip()**) rather than rotating the x-axis labels ...

```
ggplot(dat.tf,aes(x=gna,y=log(1+total.fruits))+geom_boxplot()+coord_flip()+
  theme_bw()
```

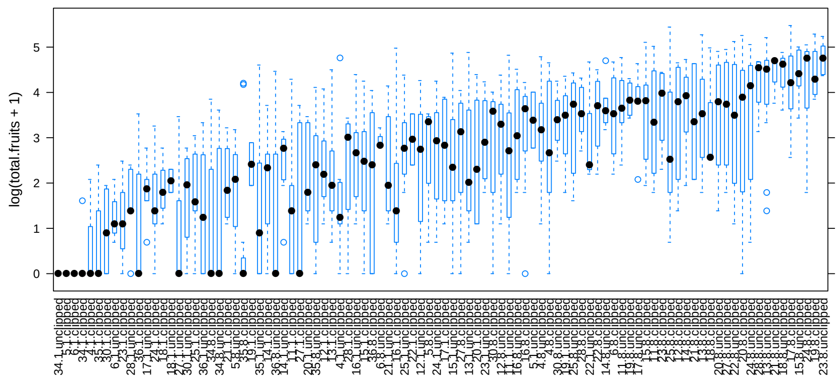


Figure 1: Box-and-whisker plots of log(total fruits+1) by treatment combinations and genotypes reveal heterogeneity of variances across groups.

Taking the logarithm of total fruit (or in this case log(1 + fruit) to avoid taking the log of zero) should stabilize the variance (McCullagh and Nelder, 1989), but this does not seem to be the case (Figure 1). The variance actually seems to decline slightly in the largest groups, but it's a bit hard to see.

We could also calculate the variance for each genotype × treatment combination and provide a statistical summary of these variances.

```
grpVarL <- with(dat.tf, tapply(log(1+total.fruits),
                               list(gna), var) )
summary(grpVarL)

##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   0.000   0.774   1.460   1.510   2.030   4.850
```

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.

Figure 1 also indicates that the mean (λ of the Poisson distribution) is well above 5 (i.e. in general the center of each group is greater than $\log(6) = 1.8$), suggesting that PQL may be appropriate, if it is the only option available. We could calculate λ for each genotype × treatment combination and provide a statistical summary of each group's λ.

```
grpMeans <- with(dat.tf, tapply(total.fruits, list(gna), mean))
summary(grpMeans)

##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##     0.0    11.4    23.2    31.9    49.7   122.0
```

The average by-group $\bar{\lambda}$ is 32; the median is ≈ 23; and at least one group has $\lambda = 0$ (i.e., complete absence of fruit set for that genotype $\times$ treatment combination).

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 proability 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, $\text{Var} = \mu(1 + \alpha\mu)$ 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 will see below how to fit data following these distributions

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

```
grpVars <- with(dat.tf,
                tapply(total.fruits,
                       list(gna), var) )  ## variance of UNTRANSFORMED data
```

[4]

We can get approximate estimates of the quasi-Poisson (linear) and negative binomial (linear/quadratic) pattern using `lm`. (The `-1` in the formulas below specifies a model with the intercept set to zero. We have to do some rearranging in the second case, and use `I()` to specify that we really mean to square the group variables

```
lm1 <- lm(grpVars~grpMeans-1)  ## `quasi-Poisson' fit
phi.fit <- coef(lm1)
lm2 <- lm((grpVars-grpMeans)~I(grpMeans^2)-1)
k.fit <- 1/coef(lm2)
## alternatively (more understandable but computationally harder):
## nls1 <- nls(grpVars~grpMeans*(1+grpMeans/k),
##    start=list(k=1))
```

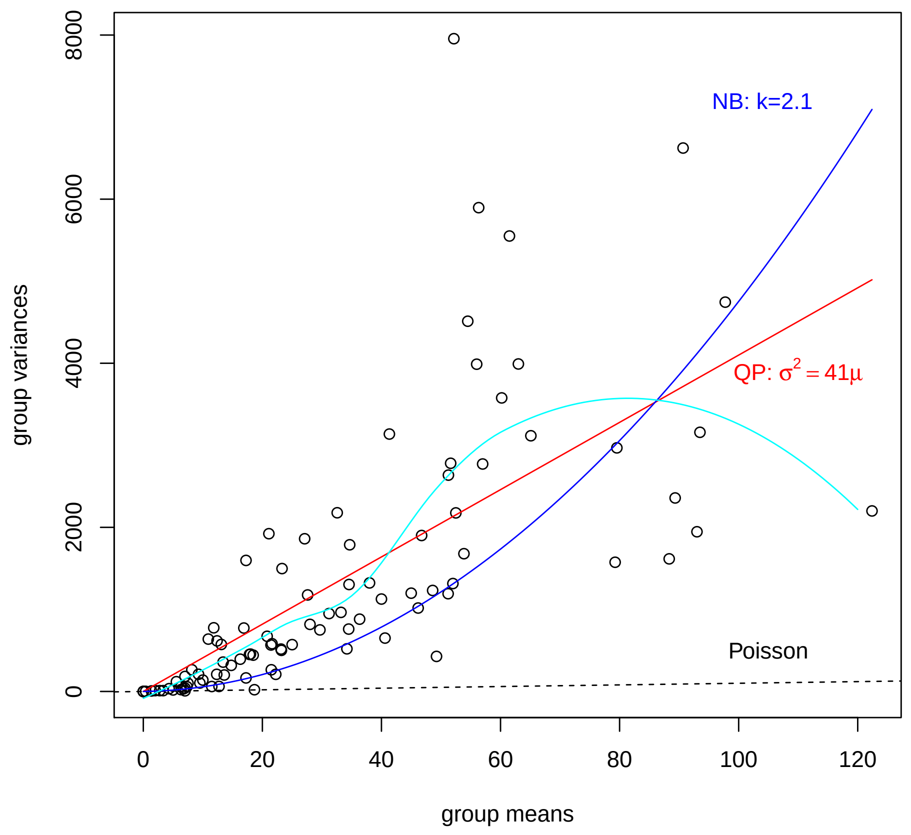
It's easy enough to plot the means vs. variances and the fitted relationships; we also add a non-parametric `loess` fit:

```
plot( grpVars ~ grpMeans, 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)
```

[4] Here are two alternative ways to compute the mean and variance of total fruits by group, using `plyr::ddply` and `reshape2::cast` (`aggregate`, from base R, works too):

```
ddply(dat.tf,"gna",
      function(x) with(x,data.frame(mean=mean(total.fruits),var=var(total.fruits))))
recast(subset(dat.tf,
              select=c(gna,total.fruits)),
       id.var=1,
       gna~...,
       fun.agg=function(x) c(mean=mean(x),var=var(x)))
```

```r
## loess fit
Lfit <- loess(grpVars~grpMeans)
mvec <- 0:120
lines(mvec,predict(Lfit,mvec),col=5)
```



5

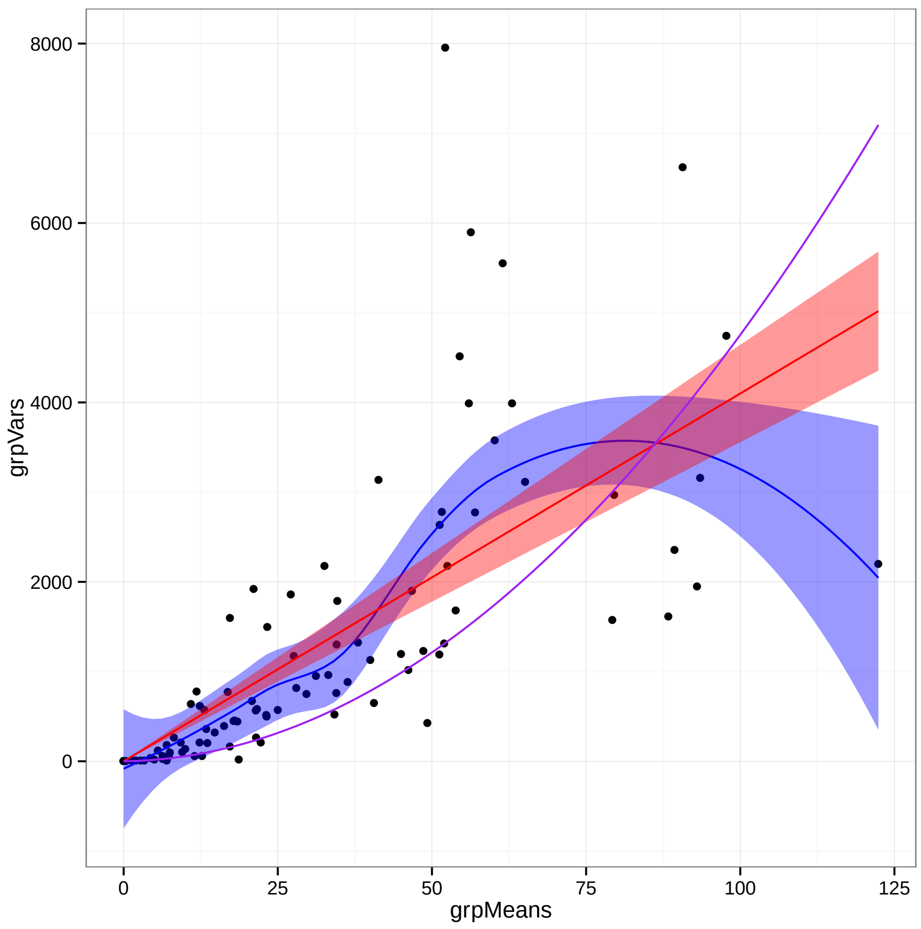
These fits are not rigorous statistical tests — they violate a variety of assumptions of linear regression (e.g. constant variance, independence), but they are good enough to give us an initial guess about what distributions we should use.

Exercise

[5] `ggplot` solution:

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

```
ggplot(data.frame(grpMeans,grpVars),
       aes(x=grpMeans,y=grpVars))+geom_point()+
  geom_smooth(colour="blue",fill="blue")+
  geom_smooth(method="lm",formula=y~x-1,colour="red",fill="red")+
  stat_function(fun=function(x) x*(1+x/k.fit),
                colour="purple",fill="purple")

## geom_smooth:  method="auto" and size of largest group is <1000, so using
loess.  Use 'method = x' to change the smoothing method.
```



- ** Draw a plot to suggest whether one might be able to stabilize the variance of the data by $\log(1 + x)$-transforming the data.

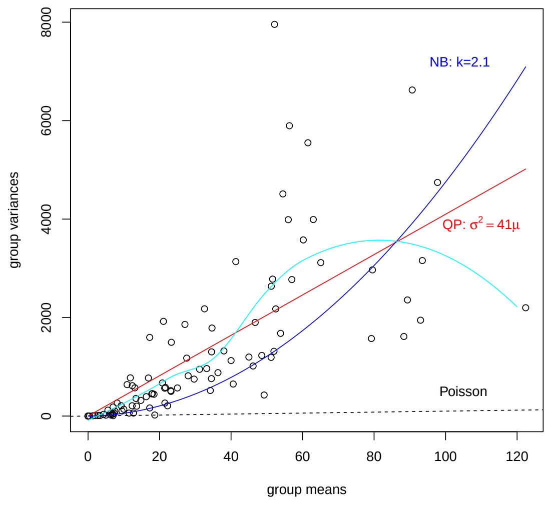


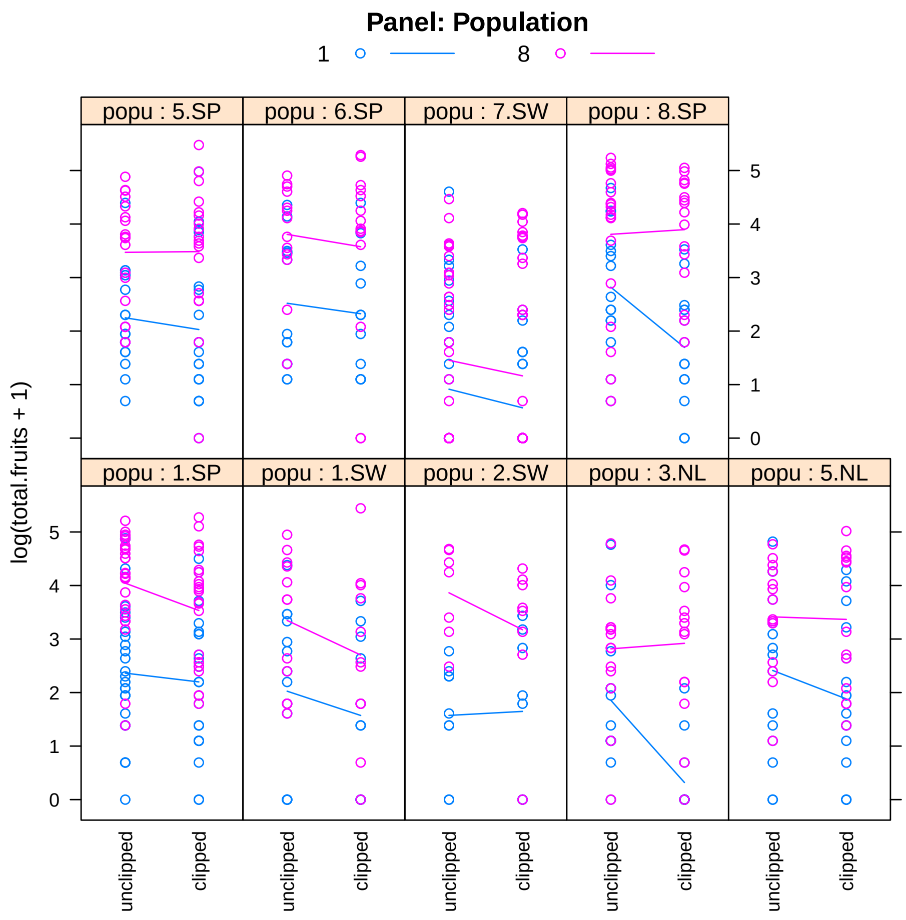
Figure 2: The variance-mean relation of Poisson random variables should exhibit a slope of approximately one (dashed line). In contrast, these data reveal a slope of much greater than one (the wedge or cornupia shape is expected for small Poisson samples, but the slope here is much, much greater than 1). The coloured lines show estimated mean-variance relationships for quasi-Poisson (or negative-binomial "type I", Hardin and Hilbe (2007)) [red]: $V = \phi\mu$; negative binomial or lognormal-Poisson (Elston et al., 2001) [blue]: $V = \mu(1 + \mu/k)$ or $V = \mu(1 + \mu \cdot (\exp(\sigma^2) - 1))$.

We find that the group variances increase with the mean much more rapidly than expected under the Poisson distribution. This indicates that we need to account for overdispersion in the model. @

4.1 Plotting the responses *vs.* treatments

We would like to plot the response by treatment effects, to make sure there are no surprises.

```
stripplot(log(total.fruits+1) ~ amd|popu, dat.tf,
                  groups=nutrient, auto.key=list(lines=TRUE, columns=2),
                  strip=strip.custom(strip.names=c(TRUE,TRUE)),
                  type=c('p','a'), layout=c(5,2,1),
                  scales=list(x=list(rot=90)), main="Panel: Population")
```



There seems to be a consistent nutrient effect (different intercepts), and perhaps a weaker clipping effect (negative slopes) — maybe even some compensation (steeper slope for zero nutrients, Fig. 3).

Because we have replicates at the genotype level, we are also able to examine the responses of genotypes (Fig. 4).

```
stripplot(log(total.fruits+1) ~ amd|nutrient, dat.tf,
                groups=gen,
                strip=strip.custom(strip.names=c(TRUE,TRUE)),
```

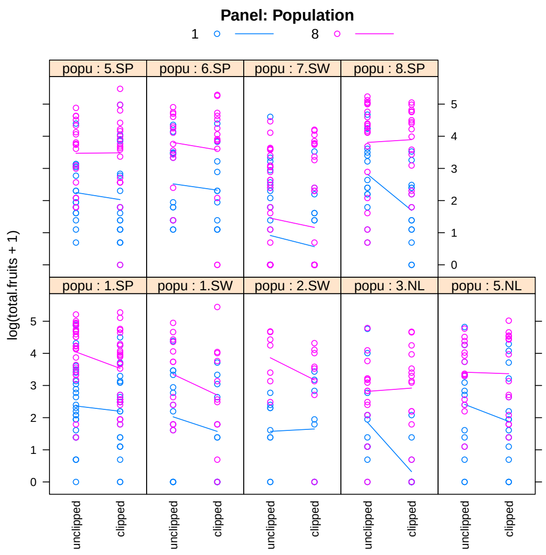


Figure 3: Interaction plots of total fruit response to nutrients and clipping, by population. Different lines connect population means of log-transformed data for different nutrient levels.

```
                 type=c('p','a'), ## points and panel-average value --
                                  ## see ?panel.xyplot
                 scales=list(x=list(rot=90)),
                 main="Panel: nutrient, Color: genotype")
```

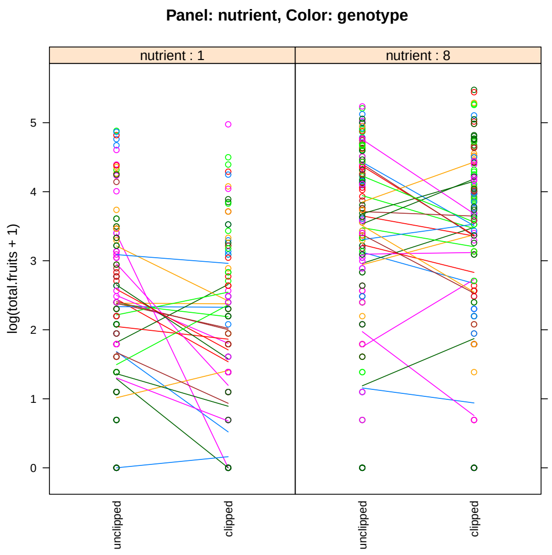
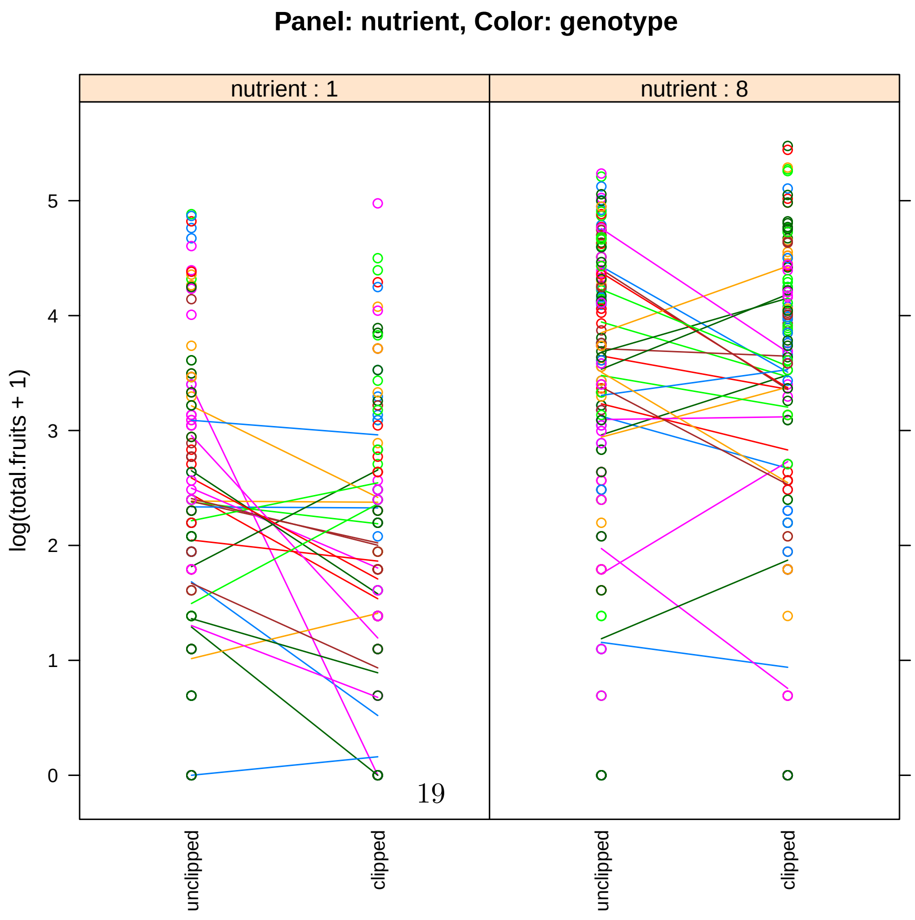


Figure 4: Interaction plots of total fruit response to clipping, for each genotype (lines and colors). Different panels show different nutrient levels.



6

There appears to be a consistent nutrient effect among genotypes (means in right panel higher than left panel), but the variation increases dramatically, obscuring the clipping effects and any possible interaction (Fig. 4).

5 Fitting group-wise GLMs

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")
```

7

There are a variety of ways to examine the different coefficients. We have written a function to save all the coefficients in a data frame, order the data frame by rank order of one of the coefficients, and then plot all the cofficients together (see (Pinheiro and Bates, 2000)); it was loaded when we `source()`ed the file `glmm_funs.R`.

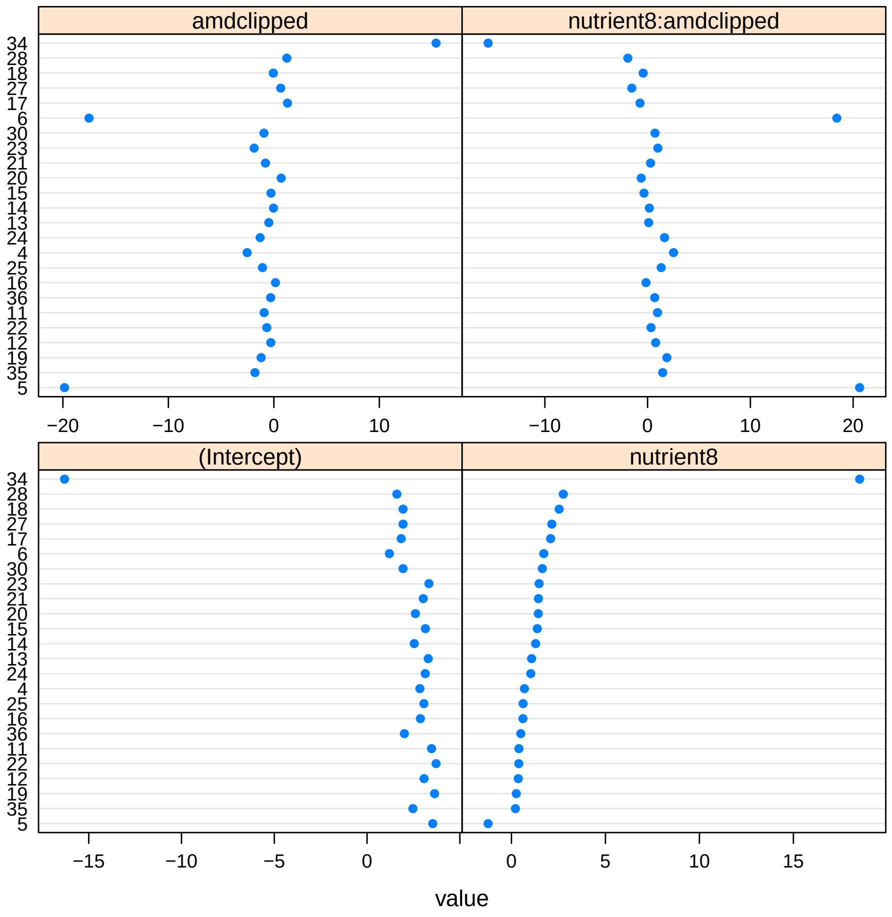
Plot the list of models created above:

[6]`ggplot` versions:

```
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.y=mean,geom="line")+
  theme_bw()+opts(panel.margin=unit(0,"lines"))+
  facet_wrap(~popu)
ggplot(dat.tf,aes(x=amd,y=log(total.fruits+1),colour=gen))+
  geom_point()+
  stat_summary(aes(x=as.numeric(amd)),fun.y=mean,geom="line")+
  theme_bw()+
  ## label_both adds variable name ('nutrient') to facet labels
  facet_grid(.~nutrient,labeller=label_both)
```

[7]The versions of `lmList` in the `lme4` and `nlme` packages are different (only the one in `lme4` has a `family` argument for running GLMs). If `find("lmList")` shows a version from `nlme`, use `detach("package:nlme")` to get rid of it (you may need to detach the upstream package that loaded it first), or use `lme4::lmList`.

```
plot(glm.lis,scale=list(x=list(relation="free")))

## Using grp as id variables
```



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 (Figure 6):

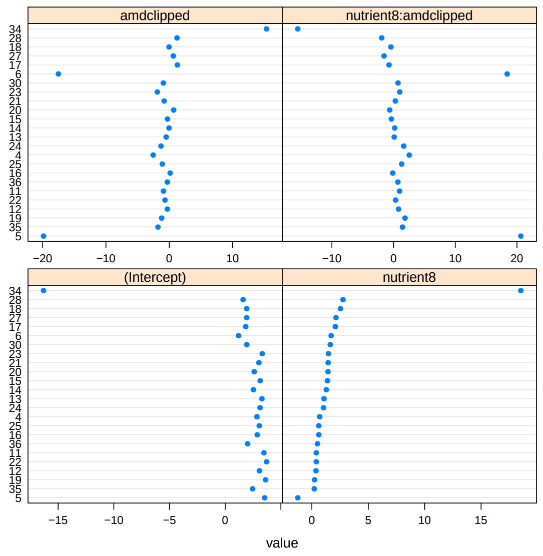
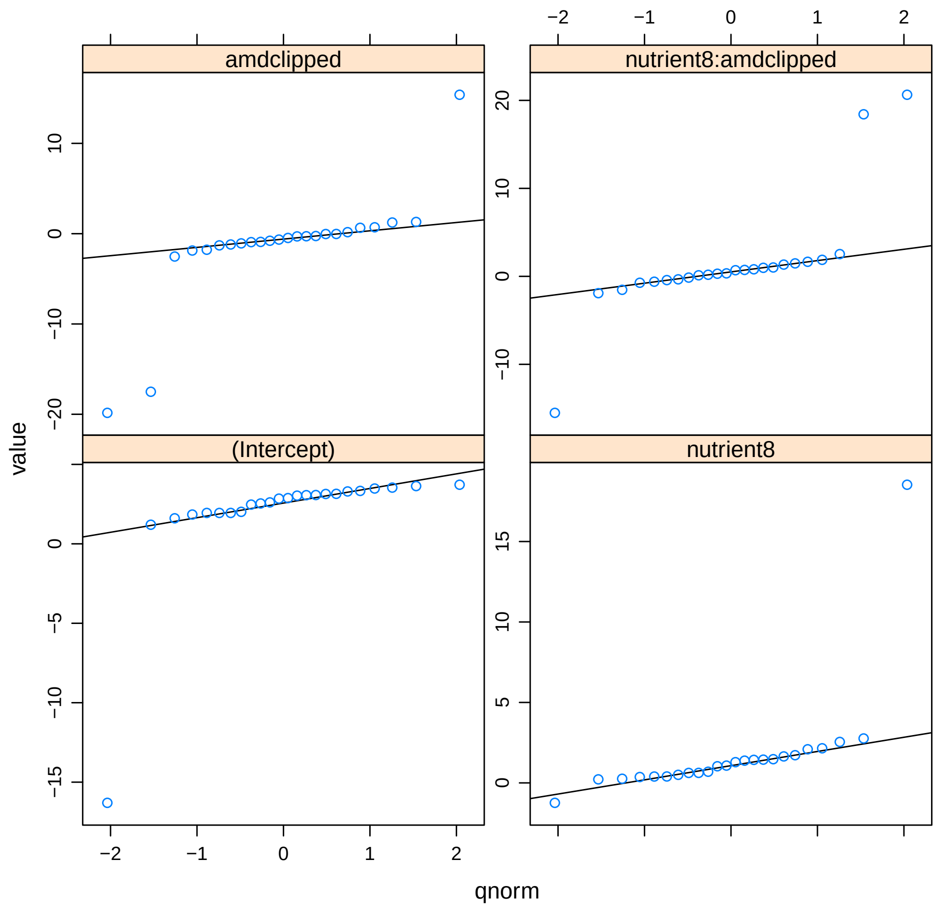


Figure 5: Model coefficients for GLM fits on each genotype.

```
qqmath(glm.lis)

## Using  as id variables
```



We see that these extreme coefficients fall far outside a normal error distribution (Fig. 6). 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. It turns out that two out of three of the outlier genotypes have small sample sizes, although the third is near the maximum sample size:

```
gentab <- with(dat.tf,table(gen))
summary(as.numeric(gentab))

##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
```

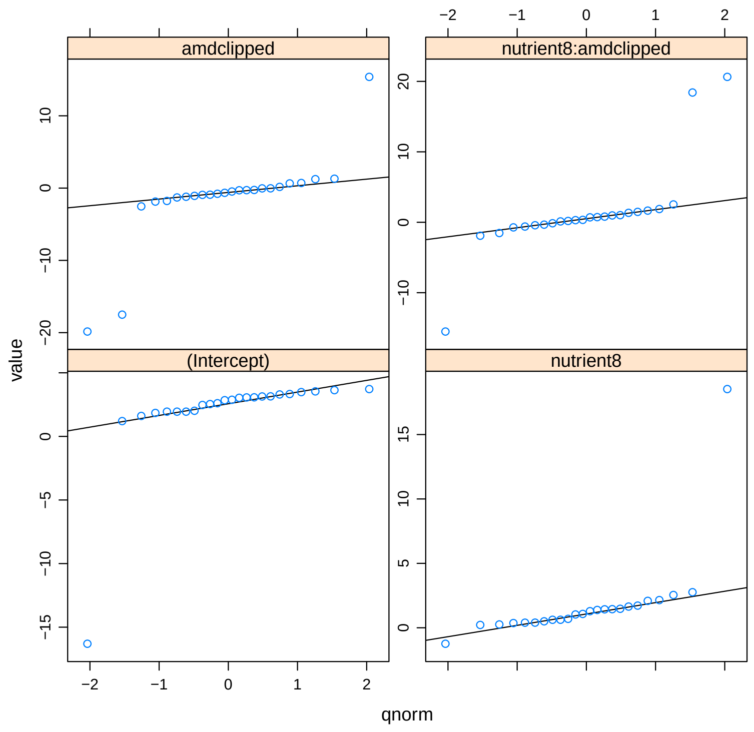


Figure 6: Q-Q plots of model coefficients for GLM fits on each genotype.

```
##       11       14       25       26       35       47

gentab[c("5","6","34")]

## gen
##  5  6 34
## 11 13 45
```

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.[8]

[8]Other options such as weakening the distributional assumptions on the random effects

6 Fitting and evaluating the full GLMM

Now we (try to) build and fit a full model, using `glmer` in the `lme4` package[9]. 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:  Model failed to converge with max|grad| = 0.00242011
(tol = 0.001, component 27)
```

This fits without any (apparent) problems — the model converges without warnings.

We can use the Pearson residuals (i.e., $r_P = (y_i - u_i)/\sqrt{Var_i}$, where the variance is λ and is therefore the fitted value for each observation) to assess overdispersion quantitatively, although it's really unnecessary in this case because the pattern (Figure 2) is so obvious: if r_{Pi} are the Pearson residuals and d is the number of residual degrees of freedom, we want $\sum r_{Pi}^2 \approx d$ — or $\sum r_{Pi}^2/d \approx 1$, or more precisely, $\sum r_{Pi}^2 \sim \chi_d^2$ (Venables and Ripley, 2002, but note warnings about applying this criterion for data with small sample means). The p-value is so incredibly low that we'll compute its logarithm instead (if we don't, R just reports 0). We define a convenience function:

```
overdisp_fun(mp1)

##    chisq    ratio        p
## 13909.46    23.26     0.00
```

(making them completely nonparametric or using a fatter-tailed distribution such as the Student t) are beyond the scope of this note, although they can be achieved via general-purpose tools such as WinBUGS or AD Model Builder.

[9] As of this writing, `glmer` and `lmer` are nearly synonymous: calling `lmer` with a `family` argument automatically hands off to `glmer` to run a GLMM. This may change in the future, and using `glmer` makes it clearer that one is really trying to run a GLMM rather than a LMM.

This shows that the data are (extremely) over-dispersed, given the model. The *deviance* of the model (in this case, a penalized version of -2 times the log-likelihood) gives a similar (also approximate for the purposes of inference)

```
deviance(mp1)

## [1] 13909
```

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:  failure to converge in 10000 evaluations
## Warning:  Model failed to converge with max|grad| = 0.0813591
(tol = 0.001, component 8)
## Warning:  Model failed to converge:  degenerate  Hessian with
1 negative eigenvalues
```

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 nutrient8 amdclipped:nutrient8
## (Intercept)                1.0000    -0.9992   -0.9886                0.8437
## amdclipped                -0.9992     1.0000    0.9906               -0.8526
## nutrient8                 -0.9886     0.9906    1.0000               -0.9141
## amdclipped:nutrient8       0.8437    -0.8526   -0.9141                1.0000
```

The `printvc` is a utility function that prints the variance-covariance matrices along with an equals sign (=) appended to any covariance component that corresponds to a perfect (or nearly perfect ± 1 correlation:

```
printvc(mp2)

## X:
```

```
##             (Intercept)
## (Intercept) 2
##
## gen:
##                       (Intercept) amdclipped nutrient8 amdclipped:nutrient8
## (Intercept)            0.269
## amdclipped            -0.079=      0.023
## nutrient8             -0.203       0.060      0.157
## amdclipped:nutrient8  0.103       -0.031     -0.085     0.055
##
## popu:
##                       (Intercept) amdclipped nutrient8 amdclipped:nutrient8
## (Intercept)            0.4862
## amdclipped             0.0671      0.2074
## nutrient8              0.0875      0.0530     0.0243
## amdclipped:nutrient8 -0.0023     -0.2319    -0.0482    0.2712
```

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:  Model failed to converge with max|grad| = 0.0208062
(tol = 0.001, component 5)
```

```
printvc(mp3)

## X:
##             (Intercept)
## (Intercept) 2
##
## gen:
##             (Intercept) amdclipped nutrient8
## (Intercept)  0.2296
## amdclipped  -0.0063      0.0007
## nutrient8   -0.1554      0.0042     0.1055
##
## popu:
##             (Intercept) amdclipped nutrient8
## (Intercept) 0.504
## amdclipped  0.075       0.011
## nutrient8   0.082=      0.012      0.013
```

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. [10]

[10] One can also model the interaction of categorical predictors (such as clipping) with a random effect in a slightly more restricted way, for example coding the `(amd|popu)`

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 at each of these levels (suggesting that we have not overfitted the model).

```
## Warning:  Model failed to converge with max|grad| = 0.00342253
(tol = 0.001, component 4)
```

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):

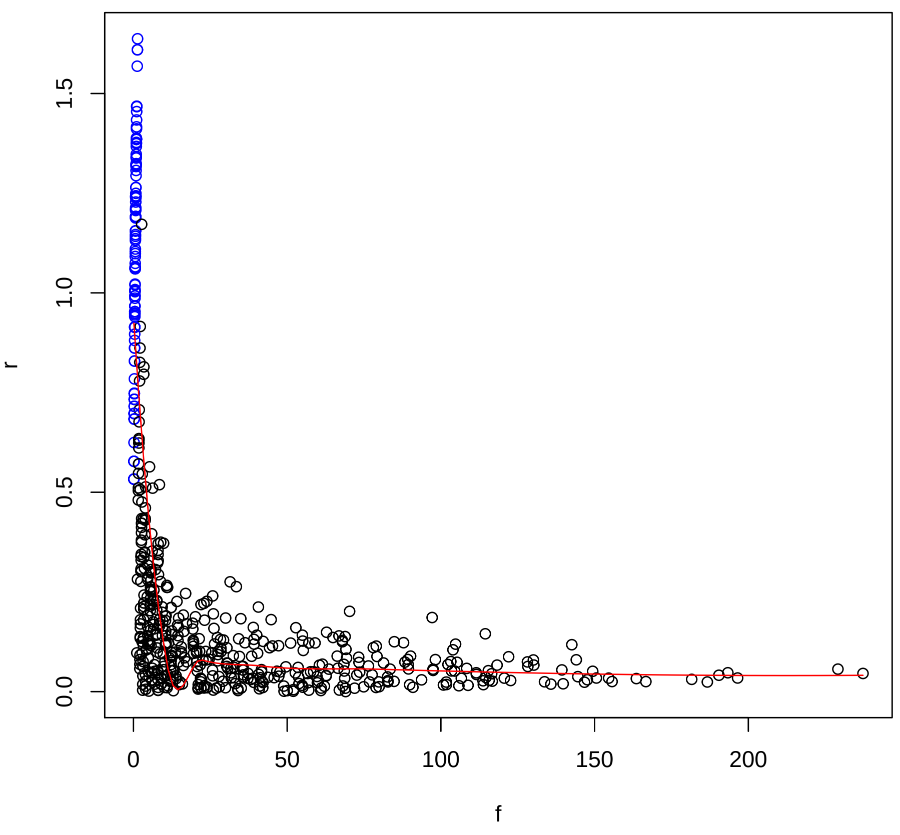
```
overdisp_fun(mp4)

##    chisq    ratio        p
## 177.4376   0.2885   1.0000
```

Looks like we're fine now in terms of the residual χ^2 criterion.

Next we apply a function from `glmm_funs.R` that defines a location-scale plot (Figure 7: this is plot #3 in the standard `plot.lm` sequence). The variance pattern is slightly alarming, with higher variance for small fitted values. Some but not all of this pattern is generated by zeros in the original data …

```
locscaleplot(mp4,col=ifelse(dat.tf$total.fruits==0,"blue","black"))
```

term in the model above as `(1|popu/amd)` instead. In effect, we are treating the different `amd` levels as nested groups within populations. This approach is slightly less flexible (for example, one cannot have negative correlations between treatment levels) but slightly easier computationally. In this case, trying to fit models with `amd` or `nutrient` nested within population or genotype leads to zero variances for the lowest level — in other words, just as we concluded before, we don't have enough power to detect variation in treatment effects among groups.



7 Inference

Now that we have a full model, we wish to make inferences regarding the parameters.

7.1 Comparing models with different random effects

We can draw a (somewhat ugly) plot of the estimated random effect standard deviations for our model (Figure 8):

```
coefplot2(mp4,ptype="vcov",intercept=TRUE,cex.pts=1.5)
```

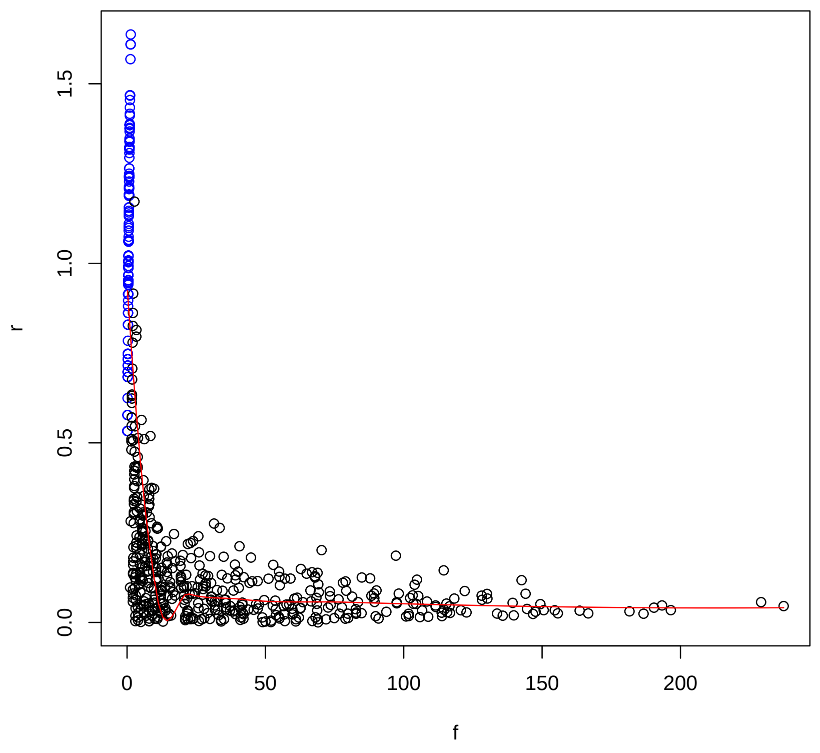
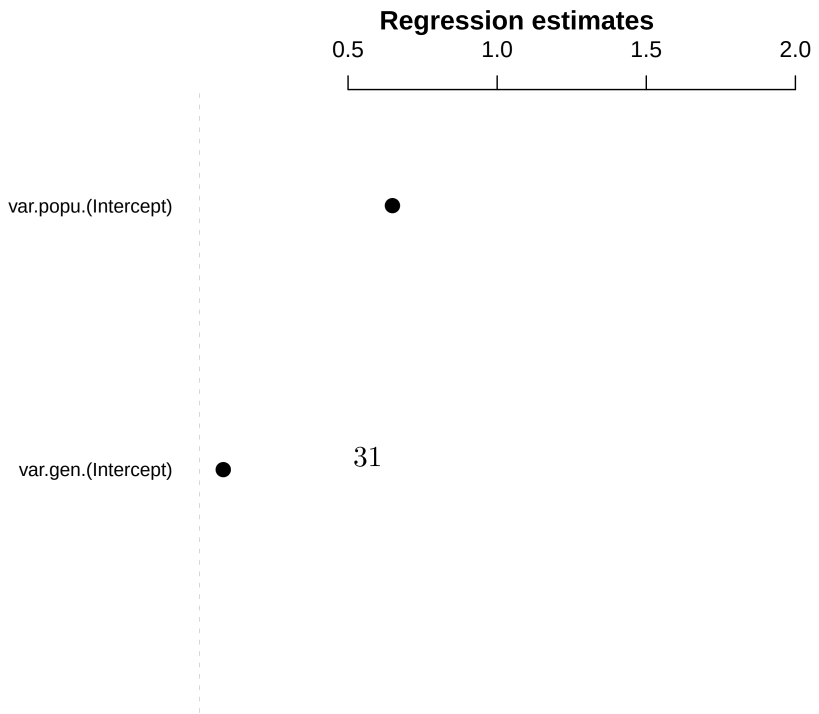


Figure 7: Location-scale plot, with superimposed loess fit. Zeros in the data are marked in blue.



glmer does not return information about the standard errors or confidence intervals of the variance components, so these values are represented only as their point estimates.

As much as I like graphical presentations, in this case I have to admit that the printed representation given by VarCorr(mp4) or printvc(mp4) would be just as useful.

Exercise Try it.

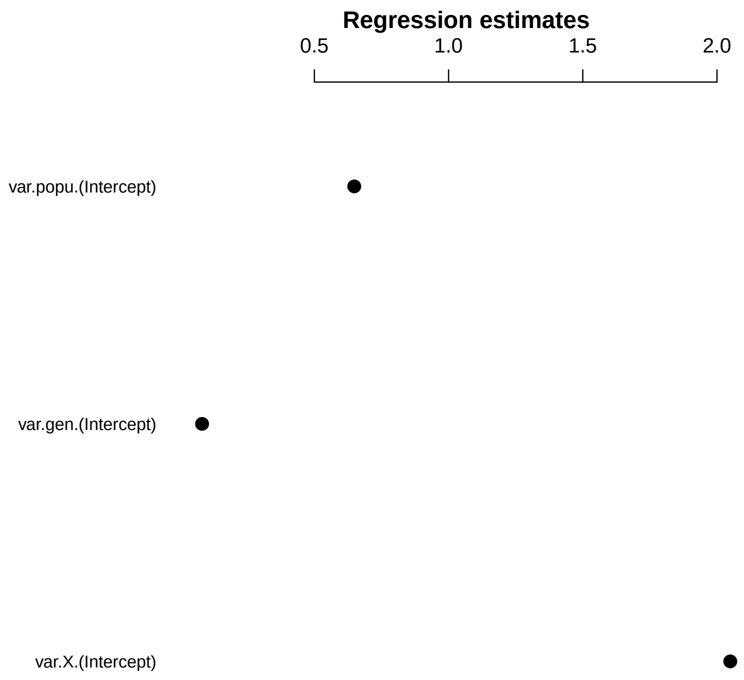


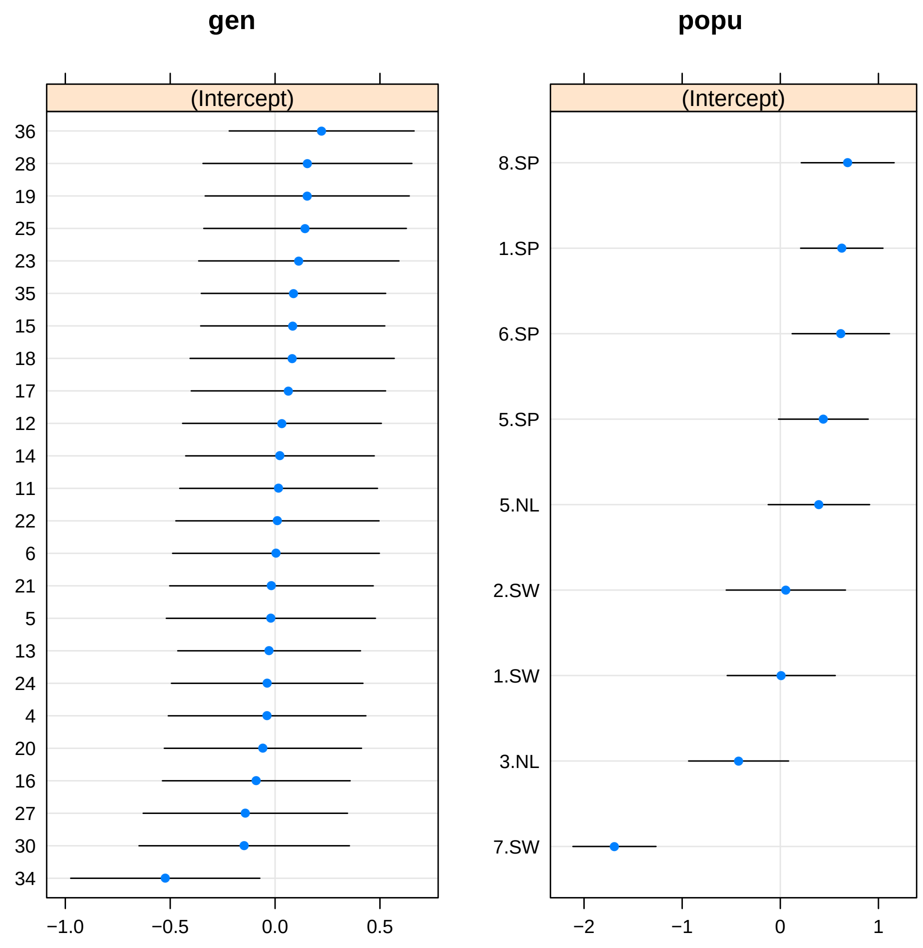
Figure 8: Estimated standard deviations for fitted model.

We can also inspect the random effects estimates themselves (in proper statistical jargon, these might be considered "predictions" rather than "estimates" (Robinson, 1991): Figure **??**). 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.

(We use a bit of trickery from the `gridExtra` package to arrange these plots on the page, and we ignore the observation-level random effects, which are returned as the `$X` component of the list of plots.)

There is a hint of regional differentiation — the Spanish populations have higher estimated/predicted fruit sets than the Swedish and Dutch populations. Genotype 34 again looks a little bit unusual.

```
## squash margins a bit ...
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 <- dotplot(r1, par.settings=pp)
print(grid.arrange(d1$gen,d1$popu,nrow=1))
```



```
## NULL
```

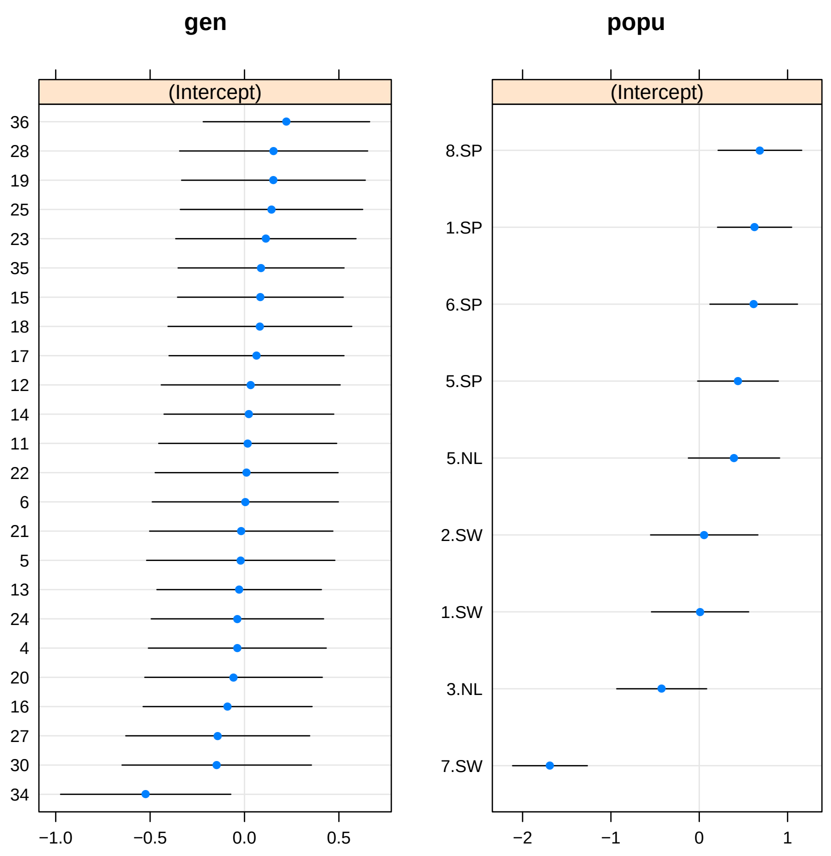
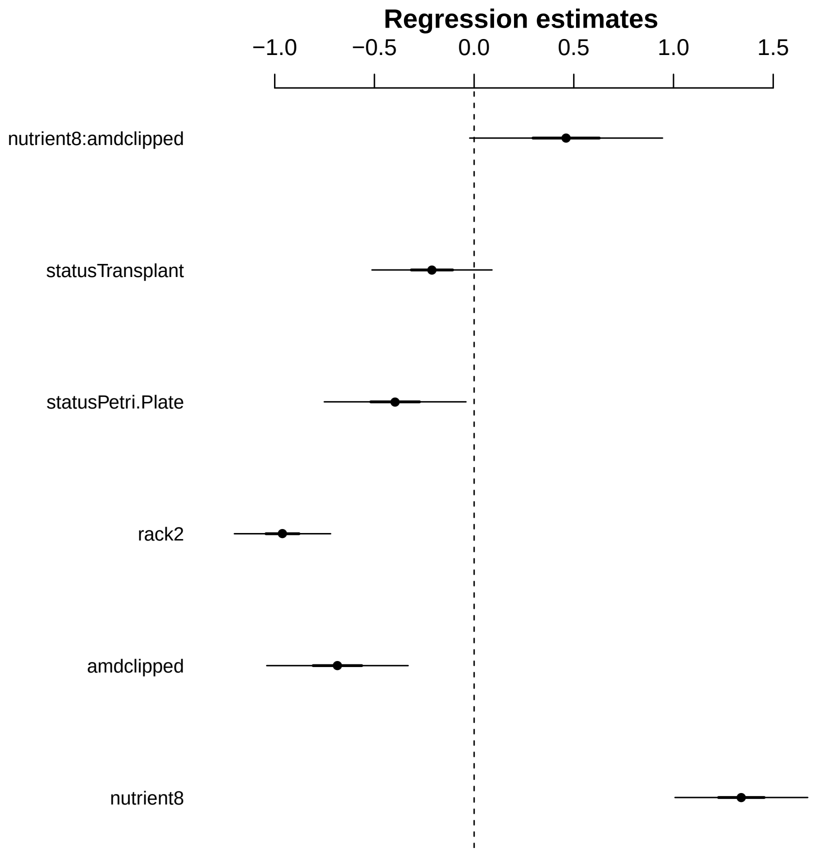


Figure 9: Random effects estimates/predictions for model `mp4`

7.2 Comparing models with different fixed effects

What are the fixed effects?

```
coefplot2(mp4)
```



`glmm_fun.R` defines a convenience function, `daic()`, that converts the AIC tables returned by `drop1` (which we will use momentarily) into ΔAIC tables:

We use the `drop1` function to assess both the AIC difference and the likelihood ratio test between models. 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.[11]

[11]In general you should *not* explore both AIC- and LRT-based comparisons when analyzing a models; they represent different inferential approaches, and you should decide before analyzing the model which you prefer, to avoid the temptation of data snooping or "cherry-picking" (i.e., comparing the results of the two approaches and choosing the ones you prefer). Here we show both analyses for illustration.

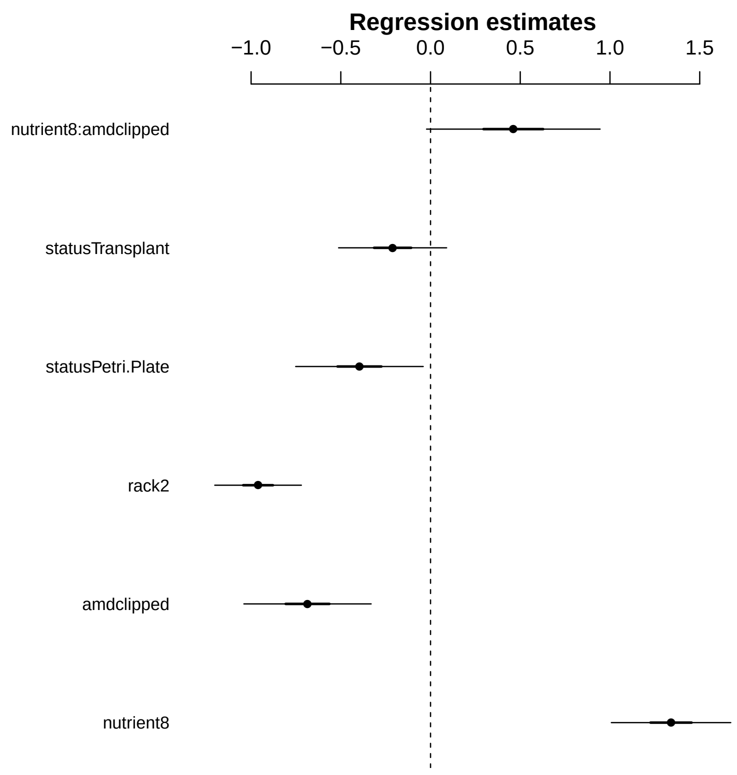


Figure 10: Fixed-effect parameter estimates (±1 (thick line) and ±2 (thin line) standard error) for the full model

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

## Single term deletions
##
## Model:
## total.fruits ~ nutrient + amd + rack + status + (1 | X) + (1 |
##     popu) + (1 | gen) + nutrient:amd
##                Df dAIC
## <none>            0.0
## rack            1 55.1
## status          2  1.6
## nutrient:amd  1  1.4
```

```
## Single term deletions
##
## Model:
## total.fruits ~ nutrient + amd + rack + status + (1 | X) + (1 |
##     popu) + (1 | gen) + nutrient:amd
##              Df dAIC
## <none>           0.0
## rack          1 55.1
## status        2  1.6
## nutrient:amd  1  1.4
```

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

```
## Single term deletions
##
## Model:
## total.fruits ~ nutrient + amd + rack + status + (1 | X) + (1 |
##     popu) + (1 | gen) + nutrient:amd
##              Df  AIC  LRT Pr(Chi)
## <none>          5015
## rack          1 5070 57.1 4.2e-14 ***
## status        2 5017  5.6   0.060 .
## nutrient:amd  1 5017  3.4   0.064 .
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
## Single term deletions
##
## Model:
## total.fruits ~ nutrient + amd + rack + status + (1 | X) + (1 |
##     popu) + (1 | gen) + nutrient:amd
##              Df  AIC  LRT Pr(Chi)
## <none>          5015
## rack          1 5070 57.1 4.2e-14 ***
## status        2 5017  5.6   0.060 .
## nutrient:amd  1 5017  3.4   0.064 .
## ---
## 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 ($\Delta\text{AIC} = 1.4$), 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. [12] (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)
anova(mp5,mp4)

## Data: dat.tf
## Models:
## mp5: total.fruits ~ nutrient + amd + rack + status + (1 | X) + (1 |
## mp5:     popu) + (1 | gen)
## mp4: total.fruits ~ nutrient + amd + rack + status + (1 | X) + (1 |
## mp4:     popu) + (1 | gen) + nutrient:amd
##     Df  AIC  BIC logLik deviance Chisq Chi Df Pr(>Chisq)
## mp5  9 5017 5057  -2499     4999
## mp4 10 5015 5060  -2498     4995  3.44      1      0.064 .
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

We an also test all the reduced models:

[12]There is significant controversy over this point. Some researchers state that centering the input variables (in this case using `contr.sum` to change to "sum to zero" contrasts) will make the main effects interpretable even in the presence of the interaction term Schielzeth (2010), while others say one should almost never violate the "principle of marginality" in this way (Venables, 1998). Dropping the interactions may be an appropriate way forward (Pinheiro and Bates, 2000), although others say that dropping non-significant terms from models is almost never justified (Harrell, 2001; Whittingham et al., 2006) . . .

```
(dd_AIC2 <- dfun(drop1(mp5)))

## Single term deletions
##
## Model:
## total.fruits ~ nutrient + amd + rack + status + (1 | X) + (1 |
##     popu) + (1 | gen)
##          Df  dAIC
## <none>        0.0
## nutrient  1 135.7
## amd       1  10.2
## rack      1  54.2
## status    2   1.3

(dd_LRT2 <- drop1(mp5,test="Chisq"))

## Single term deletions
##
## Model:
## total.fruits ~ nutrient + amd + rack + status + (1 | X) + (1 |
##     popu) + (1 | gen)
##          Df  AIC   LRT Pr(Chi)
## <none>      5017
## nutrient  1 5153 137.7 < 2e-16 ***
## amd       1 5027  12.2 0.00047 ***
## rack      1 5071  56.2 6.4e-14 ***
## status    2 5018   5.3 0.07116 .
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

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 the `doBy` package (see `www.warwick.ac.uk/statsdept/useR-2011/abstracts/290311-halekohulrich.pdf`)), but they do not apply to GLMMs.

We can do a parametric bootstrap comparison between nested models, but it is too computationally intensive to do lightly.

```
library(doBy)
pb1 <- PBrefdist(mp4,mp5)
PBmodcomp(mp4,mp5,ref=pb1)
```

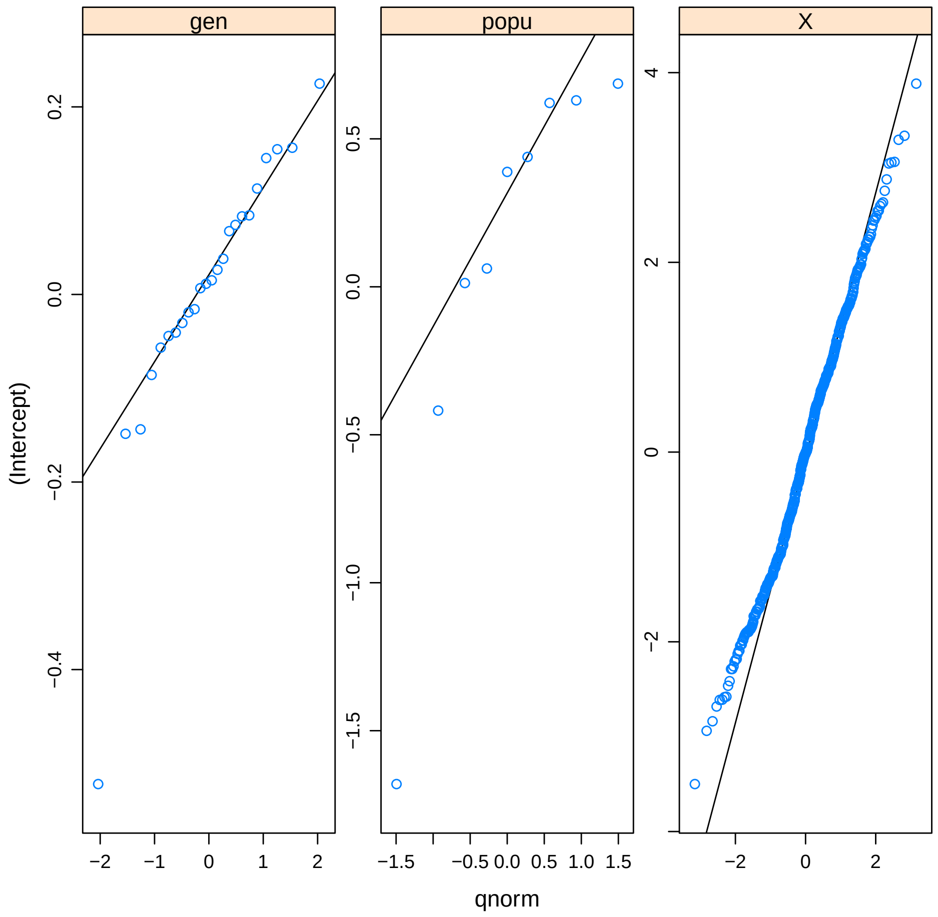
At this point, we could re-check our assumptions.

```
overdisp_fun(mp5)

##    chisq    ratio        p
## 177.2036   0.2877   1.0000
```

If we did `locscaleplot(mp5)` we would see very much the same pattern as in Figure 7.

In addition, we can check the normality of the random effects and find they are reasonable (Fig. 11). 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: see `?qqmath`.

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



8 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 the correlation between them, 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

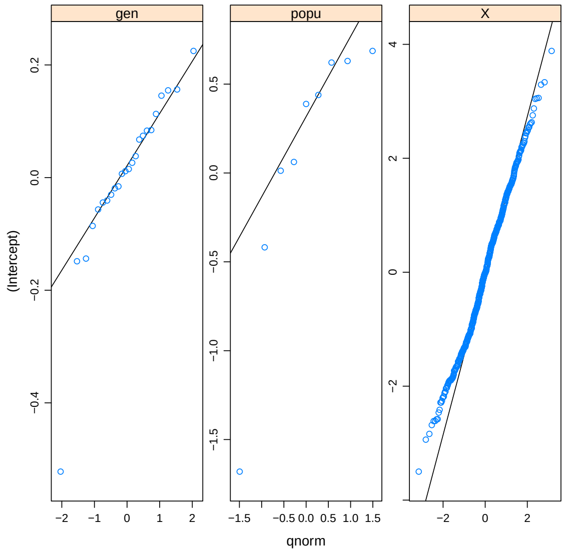


Figure 11: Quantile-quantile plot of the genotype random effects of the final model.

statement that "nutrients cancel out the effect of herbivory". To test this equivalence formally, we could set the analysis up as a one-way layout (with levels [`Nutrient1:Unclipped`, `Nutrient1:Clipped`, `Nutrient8:Clipped`, `Nutrient8:Unclipped`]); the contrast between `Nutrient1:Unclipped` and `Nutrient8:Clipped` would then test the hypothesis (see `http://glmm.wikidot.com/local--files/examples/culcita_glmm.pdf` for a similar example).

Exercise

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

9 Alternative approaches

9.1 glmmADMB

Our first alternative is the `glmmADMB` package, which uses an approach generally similar to `lme4` although its history and computational approach are very different. The main advantage of `glmmADMB` is its flexibility; its main disadvantage is its slowness (although the package is under rapid development).

Unfortunately, we have to juggle some package conflicts to load glmmADMB (and re-load coefplot2):

```
detach("package:coefplot2",unload=TRUE)
detach("package:lme4",unload=TRUE) ## VarCorr() conflict
library(glmmADMB)

##
## Attaching package:  'glmmADMB'
##
## The following object is masked from 'package:MASS':
##
##     stepAIC
##
## The following object is masked from 'package:bbmle':
##
##     stdEr
##
## The following object is masked from 'package:stats':
##
##     step

library(coefplot2)
```

glmmADMB can fit NB1 (family="nbinom1") an NB2 family="nbinom") models; it can also fit zero-inflated models (with zeroInflation=TRUE). We run a batch file that fits NB1 and NB2, with and without zero-inflation, using models such as

```
gnb2 <- glmmadmb(total.fruits ~ nutrient*amd +
                                    rack + status,
                                    random=~(amd*nutrient|gen)+
                                    (amd*nutrient|popu),
                                    data=dat.tf, family="nbinom")
```

Using glmmADMB, you can specify random effects separately as above, as in nlme, or in the same formula along with the fixed effects, as in lme4.

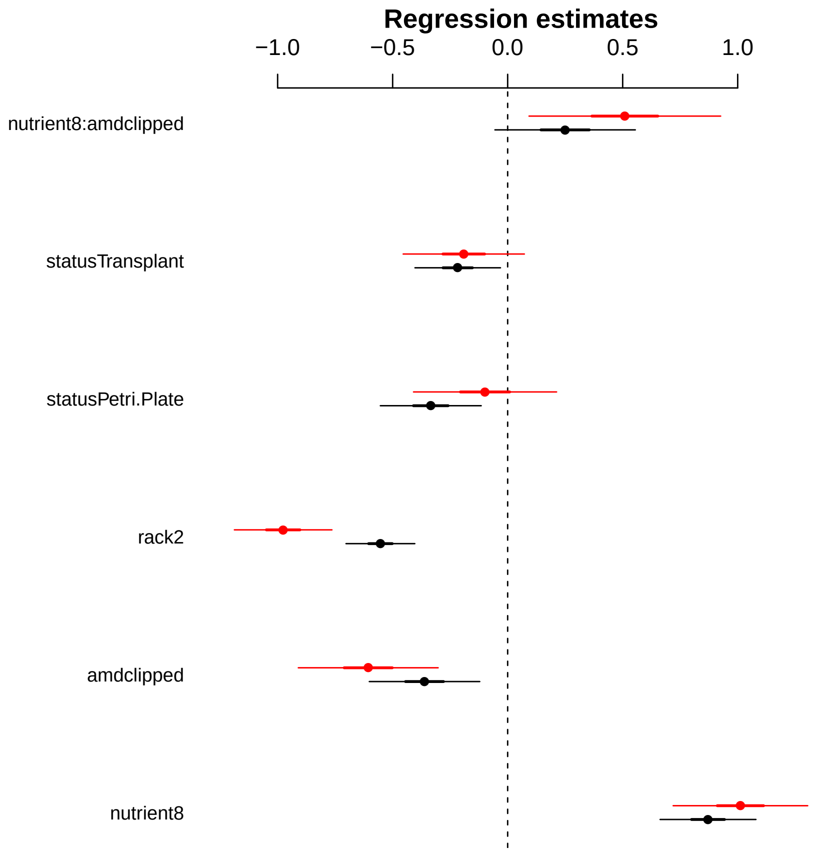
```
load("../data/Banta_glmmADMB_fits.RData")
```

The results are stored in a list called `fits`, with names `gnb[12][Z]` specifying NB type 1 or 2, without or with zero-inflation. (The `gnb1B` model will be described below.)

```
AICtab(fits)

##       dAIC  df
## gnb1C   0.0 10
## gnb1B   1.7 11
## gnb1   11.6 16
## gnb1Z  13.6 17
## gnb2  134.6 16
## gnb2Z 136.4 17
```

It looks like NB1 is *much* better than NB2 . . . but zero-inflation isn't doing anything.

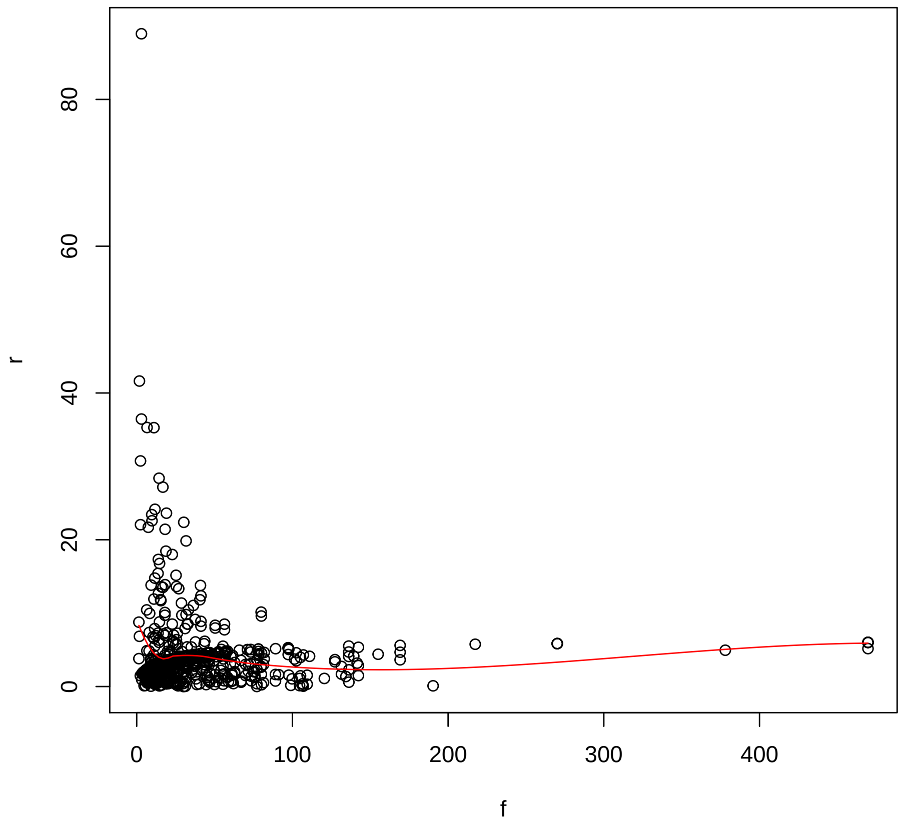
```
coefplot2(fits[c("gnb1","gnb2")])
```



The parameters are slightly different (NB2 estimates a stronger interaction effect), but not really qualitatively different — and we should go with the better-fitting model in any case.

It doesn't get rid of the artifacts in the location-scale plot, though …

```
locscaleplot(fits$gnb1)
```



By default, `glmmADMB` estimates independent variances for all of the random effects. For the population level, it estimates zero variances for everything but the intercept; for the genotype level, it estimates zero variance for clipping — but a non-zero variance for the clipping × nutrient interaction (which is a bit weird).

```
lapply(VarCorr(fits$gnb1),round,3)

## $gen
##                      (Intercept) amdclipped nutrient8 amdclipped:nutrient8
## (Intercept)                0.015          0     0.000                0.000
## amdclipped                 0.000          0     0.000                0.000
## nutrient8                  0.000          0     0.013                0.000
## amdclipped:nutrient8       0.000          0     0.000                 0.015
##
## $popu
##                       (Intercept) amdclipped nutrient8 amdclipped:nutrient8
## (Intercept)                 0.304          0         0                    0
## amdclipped                  0.000          0         0                    0
## nutrient8                   0.000          0         0                    0
## amdclipped:nutrient8        0.000          0         0                    0
```

We can instead look at the reduced model which includes only an intercept model for the population but allows variation in the effect of nutrients:

```
lapply(VarCorr(fits$gnb1B),round,3)

## $gen
##             (Intercept) nutrient8
## (Intercept)       0.014     0.000
## nutrient8         0.000     0.014
##
## $popu
##             (Intercept)
## (Intercept)       0.308
```

9.2 MCMCglmm

I don't have time to go into the details here, but you can also use Bayesian approaches . . . `MCMCglmm` is the easiest entry point into the Bayesian world, because it fits GLMMs in a fairly straightforward way (and it is very fast, by the standards of Bayesian MCMC approaches). By the way, MCMC stands for "Markov chain Monte Carlo" . . .

```
library(MCMCglmm)
```

Here's the model specification:

```
## use independent effects (idh); full covariance matrix (us) fails
m2 <- MCMCglmm(total.fruits ~ nutrient*amd +
         rack + status,
         random=~idh(amd*nutrient):popu+idh(amd*nutrient):gen,
         data=dat.tf, family="poisson",verbose=FALSE)
```

The fixed effect part should look familiar by now. The random effects part is a little bit different. This example fits the same RE model as fitted above: `idh` means to fit the various effects (clipping, nutrient, and their interaction, as specified by `amd*nutrient` independently; `~us` would instead fit the model with all possible covariances (if you try this you will see that it doesn't work ...). Other options are possible: see `MCMCglmm` or ask me. `MCMCglmm` automatically includes an observation-level random effect in all models, so we don't have to do anything explicit to account for overdispersion.

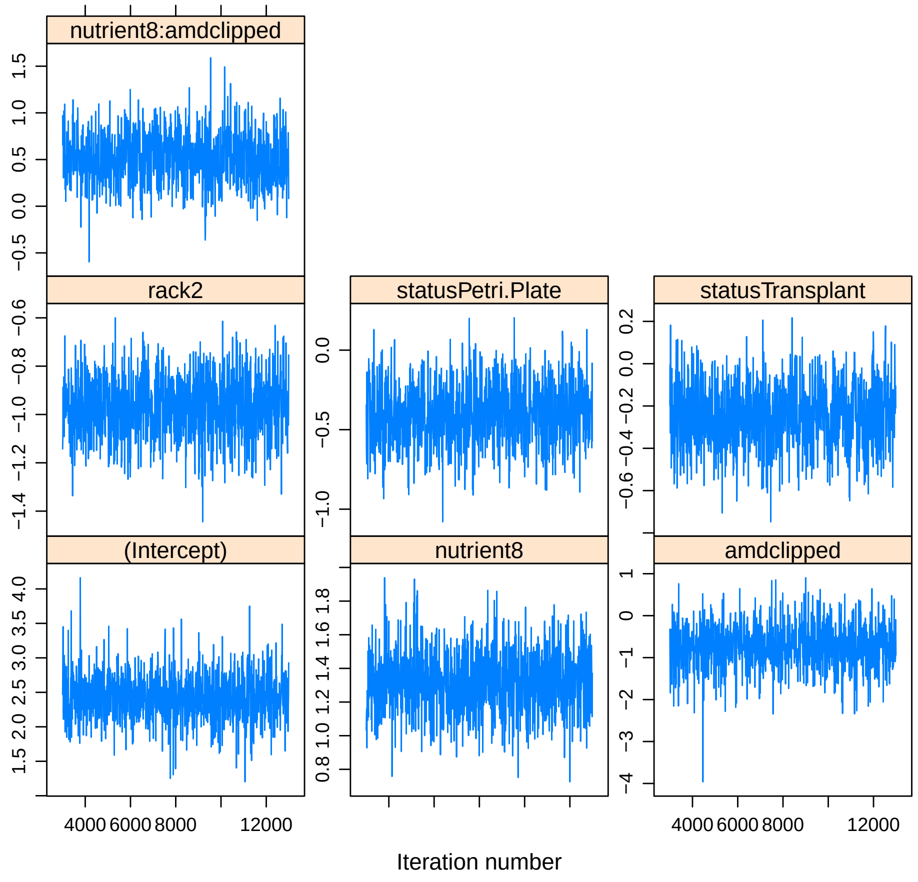
```
load("../data/Banta_MCMCglmm_fit.RData")
summary(m2)

##
##  Iterations = 3001:12991
##  Thinning interval  = 10
##  Sample size  = 1000
##
##  DIC: 3553
##
##  G-structure:  ~idh(amd * nutrient):popu
##
##                    post.mean l-95% CI u-95% CI eff.samp
## unclipped.popu       0.8288 1.27e-01    1.753    781.5
## ed.popu              1.2217 1.98e-01    2.710   1000.0
## rient8.popu          0.0208 6.54e-17    0.110    168.3
## ed:nutrient8.popu    0.0289 7.10e-17    0.185     54.9
##
##                ~idh(amd * nutrient):gen
##
##                  post.mean l-95% CI u-95% CI eff.samp
## unclipped.gen     6.92e-06 7.82e-17 1.47e-05    124.9
## ed.gen            1.71e-02 1.19e-16 1.16e-01     51.2
## rient8.gen        1.43e-03 8.95e-17 2.76e-03    104.8
## ed:nutrient8.gen  2.19e-02 8.96e-17 1.37e-01     31.8
##
##  R-structure:  ~units
```

```
##
##       post.mean l-95% CI u-95% CI eff.samp
## units      2.16     1.86     2.51      601
##
##  Location effects: total.fruits ~ nutrient * amd + rack + status
##
##                      post.mean l-95% CI u-95% CI eff.samp  pMCMC
## (Intercept)             2.4292   1.7433   3.0476     1000 <0.001 ***
## nutrient8               1.3201   0.9709   1.6885     1000 <0.001 ***
## amdclipped             -0.7468  -1.7070   0.3806     1000  0.140
## rack2                  -0.9686  -1.2410  -0.7399     1000 <0.001 ***
## statusPetri.Plate      -0.4049  -0.7842  -0.0537      845  0.034 *
## statusTransplant       -0.2453  -0.5361   0.0600      835  0.102
## nutrient8:amdclipped    0.5130   0.0517   1.0716     1000  0.050 .
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

A certain amount of extra diagnostic checking is required for MCMC models. The trace plot shows the samples taken over time. These should look like white noise — no pattern over time, and rapid variation.

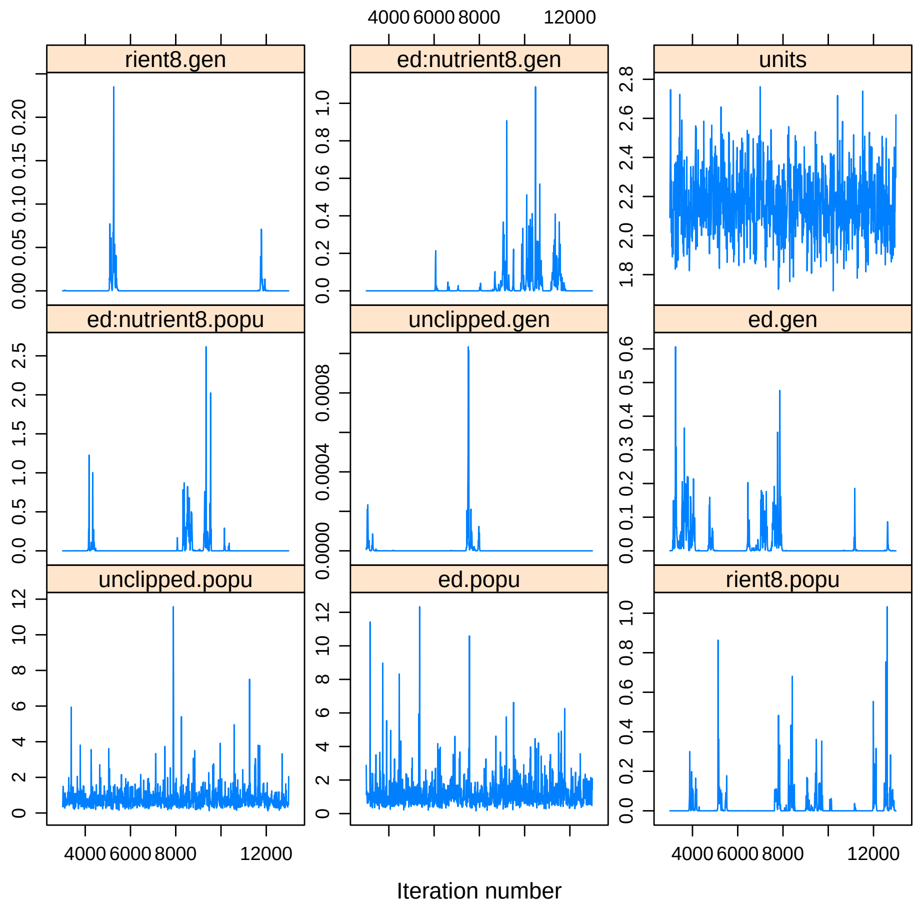
The fixed effect trace plots are an example of what trace plots *should* look like:

```
xyplot(as.mcmc(m2$Sol),layout=c(3,3))
```



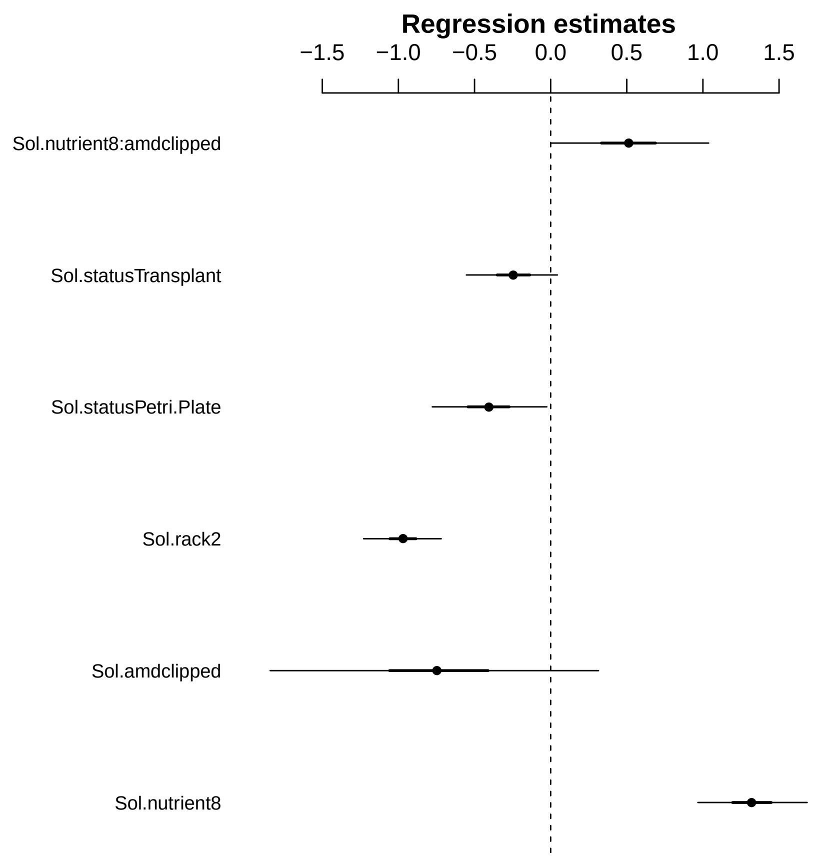
The variance parameters are an example of what the trace plots should *not* look like (except for the `units` variable, which is the observation-level random effect).

```
print(xyplot(as.mcmc(m2$VCV),layout=c(3,3)))
```

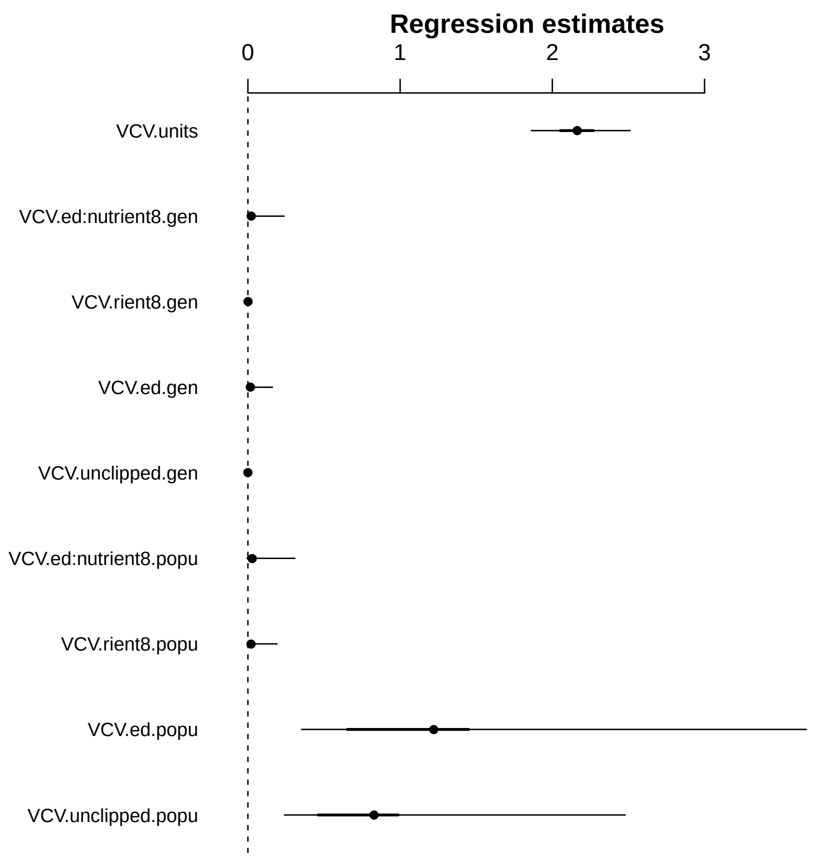


In order to get reliable answers we should discard some of the variance terms. However, for now we'll just quickly take a look at graphical output for the fixed effects and random effect variances:

```
coefplot2(m2)
```



```
coefplot2(m2,ptype="vcov",intercept=TRUE)
```



9.3 via LMM approximation

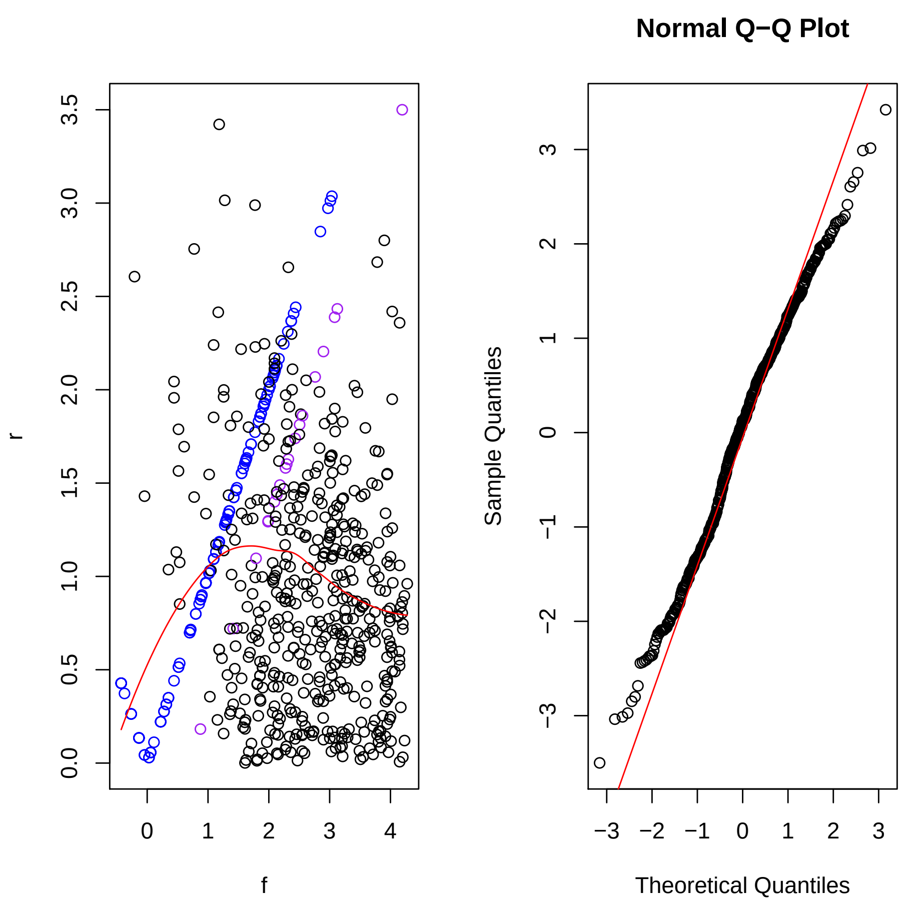
```
detach("package:glmmADMB")
library(lme4)
```

After all that effort, would we have come to a qualitatively different answer in the "usual way", transforming the data to be approximately normal and using LMMs rather than GLMMs?

```
lm1 <- lmer(log(1+total.fruits)~nutrient*amd+rack+status+(1|popu) + (1|gen),data=dat.
```

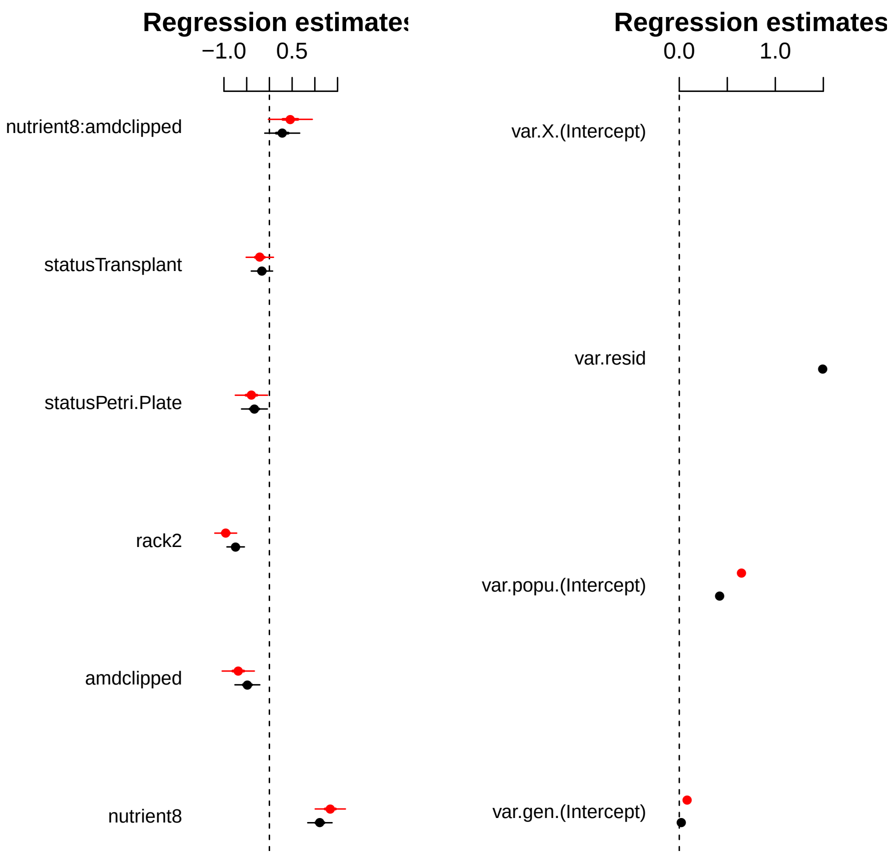
The $\log(1+x)$ transform does a reasonably good job stabilizing the variance of the residuals (although the zeros in the data set still generate a funny artifact, with the ones creating a weaker but similar pattern), although the data are still not normal (thin-tailed, rather than fat-tailed; the values at the low end of the distribution are higher than expected and *vice versa*).

```
op <- par(mfrow=c(1,2))
locscaleplot(lm1,col=ifelse(dat.tf$total.fruits==0,"blue",
                     ifelse(dat.tf$total.fruits==1,"purple","black")))

r <- residuals(lm1)
qqnorm(r)
qqline(r,col=2)
```



```
par(op)  ## restore Original Parameters
```

```
op <- par(mfrow=c(1,2))
coefplot2(list(lm1,mp4))
coefplot2(list(lm1,mp4),ptype="vcov",intercept=TRUE,xlim=c(0,1.5))

## Warning:  extra argument 'ptype' will be disregarded
```



```
par(op)
```

Experimenting with more complex RE models leads to similar conclusions . . .

References

Bolker, B. M., M. E. Brooks, C. J. Clark, S. W. Geange, J. R. Poulsen, M. H. H. Stevens, and J. S. White (2009). Generalized linear mixed models: a practical guide for ecology and evolution. *Trends in Ecology & Evolution 24*, 127–135.

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*(5), 563–569.

Hardin, J. W. and J. Hilbe (2007, February). *Generalized linear models and extensions.* Stata Press.

Harrell, F. (2001). *Regression Modeling Strategies.* Springer.

McCullagh, P. and J. A. Nelder (1989). *Generalized Linear Models.* London: Chapman and Hall.

Pinheiro, J. C. and D. M. Bates (2000). *Mixed-effects models in S and S-PLUS.* New York: Springer.

Robinson, G. K. (1991, February). That BLUP is a good thing: The estimation of random effects. *Statistical Science 6*(1), 15–32.

Schielzeth, H. (2010). Simple means to improve the interpretability of regression coefficients. *Methods in Ecology and Evolution 1*, 103–113.

Venables, W. and B. D. Ripley (2002). *Modern Applied Statistics with S* (4th ed.). New York: Springer.

Venables, W. N. (1998). Exegeses on linear models. 1998 International S-PLUS User Conference, Washington, DC.

Whittingham, M. J., P. A. Stephens, R. B. Bradbury, and R. P. Freckleton (2006). Why do we still use stepwise modelling in ecology and behaviour? *Journal of Animal Ecology 75*(5), 1182–1189.