Lecture 6. Generalized linear models for count data.

So far we've been accumulating the building blocks required to use statistical models where the classic linear / normal assumptions don't work. Probability distributions other than the normal; nonlinear functions; and maximum likelihood methods for model fitting and inference. Now we're ready to put these together and start using them.

The term "generalized linear models" (GLMs) refers to a class of models that have certain similarities. Rather than focus on the abstract definition of GLMs, let's start with the concrete example of models using the Poisson distribution. I've already used several examples where we want to model count data, where the mean count is affected by some continuous predictor or varies between groups or both. The standard way we use the Poisson is like this:

$$\mu_i = \exp(\beta_0 + \beta_1 * X_{1i} + \beta_2 * X_{2i} + \cdots)$$

$$Y_i \sim \text{Poisson}(\mu_i)$$

Where each sample Y_i is Poisson-distributed with a mean of μ_i, and the expected value μ_i is a function of various predictors X that have coefficients β. It's important to note that *the part inside the exponential function is a linear model*. We could also write the deterministic model like this:

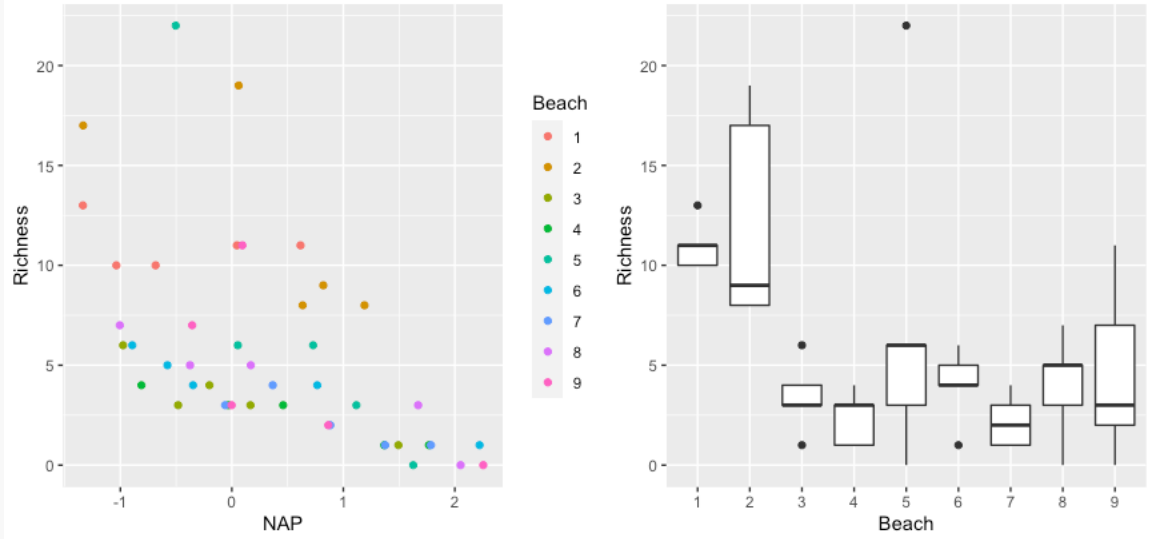
$$\log(\mu_i) = \beta_0 + \beta_1 * X_{1i} + \beta_2 * X_{2i} + \cdots$$

Because if we take the log of both sides, the exponential function on the right disappears, because log(exp(x)) = x. This way of writing the model emphasizes that the model is effectively still a linear model, if we look at the response variable on a log scale. In the GLM jargon the logarithm is the *link function* for this model, because it links the linear model to the expected value that it is predicting (μ_i). I prefer to write the model with the exponential function, because that's the way you need to write it if you want to plot the fitted model, but both ways are common and you will need to be familiar with what a link function means.

Let's look at an example analysis to see how we do a Poisson glm in R. I'll use a dataset called RIKZ where benthic species (polychaetes, crustaceans, molluscs, etc) in sandy intertidal areas were counted at 9 beaches (5 samples per beach). Instead of looking at the counts of individual species, let's look at species richness, which is the total number of species found in a sample. Species richness can be modeled with a Poisson distribution, because it is another kind of counting, and in this case the Poisson works quite well.

In the intertidal zone the gradient from low to high tidal height has strong effects on the organisms living there, as it effects duration of immersion/dessication etc. Let's look at whether species richness varies across the tidal gradient using the predictor NAP, which measure the height of a sample (meters) relative to mean tidal height.

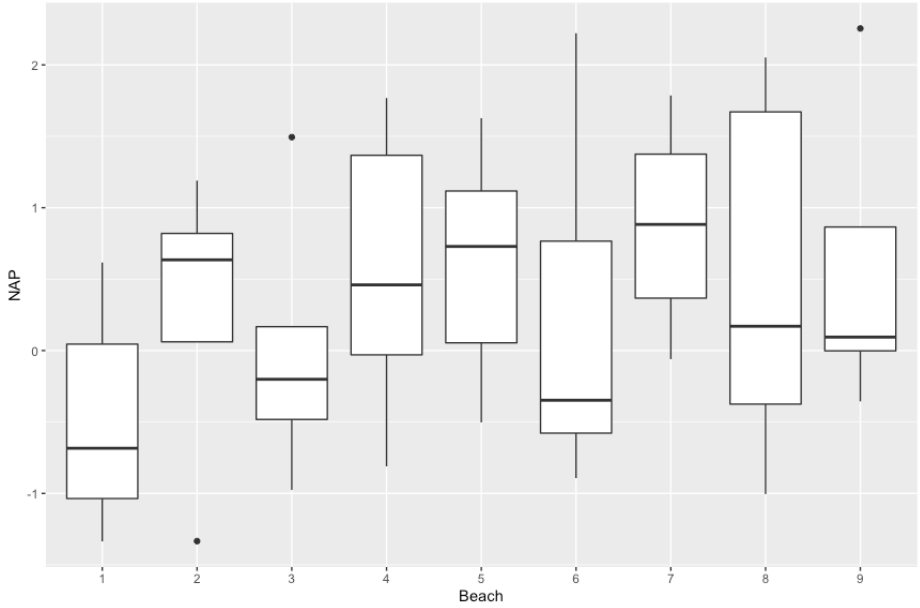
```
#read in the data, calculate richness, make Beach a factor
#read in the data, calculate richness, make Beach a factor
RIKZ <- read_table(file = "/Users/orbistertius/Documents/stats_books/zuur_analysing_ecological_data/RChapter4/RIKZ.txt")

RIKZ = RIKZ %>% mutate(Richness = rowSums(across(C1:I5, ~.x > 0)))
RIKZ$Richness <- rowSums(RIKZ[,2:76] > 0)
RIKZ$Beach = as.factor(RIKZ$Beach)

#some exploratory plots
grid.arrange(ggplot(RIKZ) + geom_point(aes(NAP, Richness, col = Beach)), ggplot(RIKZ) + geom_boxplot(aes(Beach, Richness)), nrow = 1, widths = c(1.1,1), respect = TRUE)
```



It looks like richness declines with increasing NAP. It also looks like richness varies across beaches. If we want to properly test for an effect of NAP, it's a good idea to account for Beach as well, for two important reasons. First, because multiple samples were taken per beach, it is likely that the response variable (richness) will be similar for the samples from the same beach (this is indeed what the exploratory

plot suggests). This means that if we made a model with just NAP, the residuals would be clustered by Beach, which violates the assumption of independently drawn residual error. This could result in incorrect coefficient estimates / confidence intervals / p-values. Secondly, variation in NAP may be partially confounded with variation across beaches. This is suggested by another plot:

```
ggplot(RIKZ, aes(Beach, NAP)) + geom_boxplot()
```



It does look like the tidal height at which the sample was collected does vary somewhat by beach. If we want to isolate the effect of NAP, then by controlling for beach we control for environmental variation between beaches that is confounded with variation in NAP between beaches.

Here's a model that uses NAP and beach as a predictor:

```
model = glm(Richness ~ NAP + Beach, family = poisson, data = RIKZ)
```

Specification of a model formula with glm() is the same as for lm(). We just add mulitple terms, and if we want to test for an interaction between terms we multiply them with an asterisk (here we won't explore an interaction, as there were only 5 samples per beach). The new part of the model statement is "family = poisson". The family argument is how the probability distribution of the data is specified. When poisson is specified, the default link function is the log link, which was described above. This statement could also be written 'family = poisson(link = "log")'. It is possible to specify other link functions, which will be covered later, but for poisson you almost always want the default log link.

We can explore the model fit with summary():

```
summary(model)
```

```
##
## Call:
## glm(formula = Richness ~ NAP + Beach, family = poisson, data = RIKZ)
##
## Deviance Residuals:
##    Min      1Q  Median      3Q     Max
## -2.885  -0.524  -0.119   0.381   2.604
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept)   2.1017     0.1454   14.46  < 2e-16 ***
## NAP          -0.4947     0.0764   -6.47  9.6e-11 ***
## Beach2        0.4288     0.1904    2.25  0.02431 *
## Beach3       -0.9535     0.2794   -3.41  0.00064 ***
## Beach4       -1.0591     0.3251   -3.26  0.00112 **
## Beach5        0.1290     0.2268    0.57  0.56968
## Beach6       -0.7336     0.2634   -2.79  0.00535 **
## Beach7       -0.9364     0.3464   -2.70  0.00687 **
## Beach8       -0.6248     0.2652   -2.36  0.01847 *
## Beach9       -0.3839     0.2586   -1.48  0.13765
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for poisson family taken to be 1)
##
##     Null deviance: 179.753  on 44  degrees of freedom
## Residual deviance:  47.073  on 35  degrees of freedom
## AIC: 209.1
##
## Number of Fisher Scoring iterations: 5
```

For now let's focus on the Coefficients part. There is an Intercept, a coefficent for NAP, and 8 coefficients for Beach. Referring back to our knowledge of how lm() works, we can interpet this to mean that (Intercept) is the mean richness at Beach 1 when NAP = 0. NAP is the slope for how richness changes with NAP. The other Beach coefficients quantify how each beach varies in mean richness relative to Beach 1, which is the baseline.

Recall how the log link function works:

$$\log(\mu_i) = \beta_0 + \beta_1 * X_{1i} + \beta_2 * X_{2i} + \cdots$$

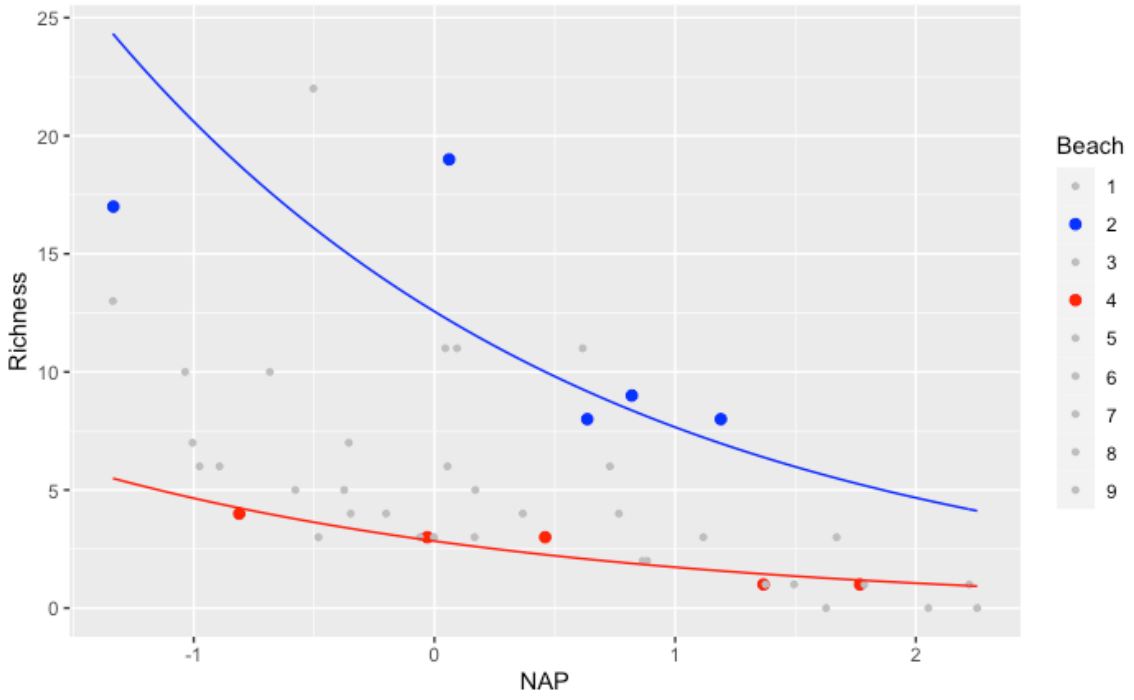
$$Y_i \sim \text{Poisson}(\mu_i)$$

This means that all the coefficients of the model (the betas) have linear effects on the log scale, or equivalently that all the coefficients are parameters in an exponential function:

$$\mu_i = \exp(\beta_0 + \beta_1 * X_{1i} + \beta_2 * X_{2i} + \cdots)$$

Therefore, a coefficient estimate of -0.49 for NAP means that species richness declines exponentially towards zero as NAP increases, at an exponential rate of -0.49 per meter. The model structure also has the consequence that any particular pair of beaches do not differ by a constant amount. *Rather, a pair of beaches will differ by a constant proportion*. We can see this by using the fitted coefficients to plot richness vs NAP at two beaches:

```
ggplot(RIKZ) + geom_point(aes(NAP, Richness, col = Beach, size = Beac
h)) +
scale_color_manual(breaks = c('1', '2', '3', '4', '5', '6', '7', '8', '
9'), values = c('grey','blue', 'grey','red', 'grey', 'grey', 'grey', 'g
rey', 'grey')) +
scale_size_manual(breaks = c('1', '2', '3', '4', '5', '6', '7', '8', '9
'), values = c(1,2,1,2,1,1,1,1,1)) +
geom_function(fun = ~exp(coef(model)["(Intercept)"]+coef(model)["Beach2
"]+coef(model)["NAP"]*.x), col = 'blue') +
geom_function(fun = ~exp(coef(model)["(Intercept)"]+coef(model)["Beach4
"]+coef(model)["NAP"]*.x), col = 'red')
```

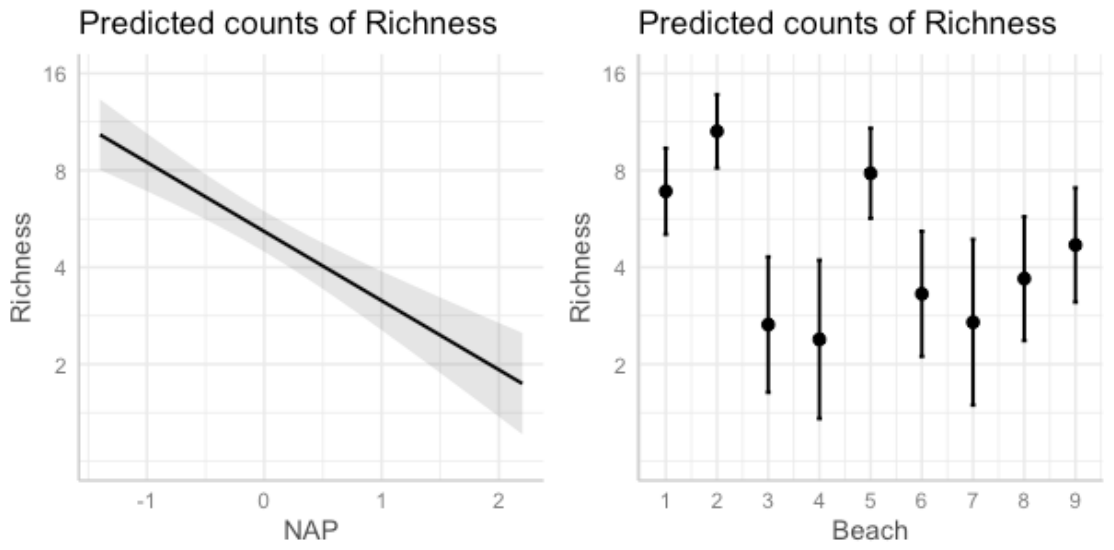


On top of the whole dataset I've plotted the samples from Beach 2 in blue and the samples from Beach 4 in red. And I've plotted what the model predicts for predicted richness at each beach. First of all, notice that the exponential model is a

good fit for this data, and this is intuitive because richness is bounded below by zero. The blue and red lines are both exponential relationships, *with the same slope but different intercepts*. We can see that when NAP is low, the difference between the two beaches is larger than when NAP is high. However, the height of these two curves always differs by the same proportion. What is this proportion? When NAP=0, the model predicts that richness at Beach 2 is exp(Intercept+Beach2) = exp(Intercept)*exp(Beach2), and that richness at Beach 4 is exp(Intercept+Beach4) = exp(Intercept)*exp(Beach4). Therefore, Beach 4 and Beach 2 differ by a factor of $\frac{e^{Beach4}}{e^{Beach2}} = e^{Beach4-Beach2}$. So at each point along the red curve, if you multiply it by exp(Beach4 – Beach2), you will get the blue curve. This would not be the case it there was an interaction in the model between NAP and Beach; then each Beach would have its own intercept and slope.

I like to plot fitted curves against raw data when possible, so that's why we need to keep track of the various nonlinear relationships assumed by the model. However, for looking at the fitted effects at a glance it can be a bit easier to interpret them on the log (link) scale, which is what the ggeffects package will return with log.y = TRUE:

```
library(ggeffects)
ggeff = ggeffect(model) %>% plot(log.y = TRUE)
grid.arrange(ggeff[[1]], ggeff[[2]], nrow = 1, respect = TRUE)
```



On the left is the fitted effect of NAP plotted on the link scale, i.e. on the log scale. One the right is the model-fitted mean richness at each beach, again on the log scale. Recall that the ggeffects package calculates (by default) the effect of NAP, averaging over Beaches, and it calculates the predicted mean count at each beach, averaging over values of NAP.

We can readily get confidence intervals for the model coefficients using confint():

```
confint(model)
```

```
## Waiting for profiling to be done...
```

```
##                 2.5 %  97.5 %
## (Intercept)  1.80545  2.3761
## NAP         -0.64636 -0.3465
## Beach2       0.05553  0.8038
## Beach3      -1.53058 -0.4282
## Beach4      -1.74248 -0.4570
## Beach5      -0.32189  0.5699
## Beach6      -1.27303 -0.2346
## Beach7      -1.66424 -0.2938
## Beach8      -1.16756 -0.1222
## Beach9      -0.90810  0.1103
```

These are calculated using the likelihood profile described earlier this lecture.

The effects of NAP and Beach look pretty strong, but let's do a real hypothesis test on those terms. glm() fits models using maximum likelihood, so we want to test these terms using a likelihood ratio test. Furthermore, we want to do a *marginal* test, i.e. I would like to test the effect of NAP while accounting for any effect of Beach, and vice versa. That means we want to 1) compare the full model to a model with NAP removed (the restricted model), and 2) compare the full model to a model with Beach removed. There are several ways to do this in R, but an easy one is to use the Anova() function in the "car" package:

```
Anova(model)
```

```
## Analysis of Deviance Table (Type II tests)
##
## Response: Richness
##       LR Chisq  Df   Pr(>Chisq)
## NAP       44.7  1    2.3e-11 ***
## Beach     66.1  8    2.9e-11 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

This function, under the default settings, will take a glm model and do marginal likelihood ratio tests on all the terms in the model. For each term it shows the Chi-squared statistic used [which is 2*log(likelihood ratio)], the degrees of freedom for the chi-squared distribution (the difference in the number of parameters between the full and restricted model), and the corresponding p-value. Both of the tests are highly significant. Note that the Beach test has Df=8, because the Beach term in the model requires fitting 8 extra parameters.

Let's inspect the output of this model a little more. Why does a function called Anova() do likelihood ratio tests? As I mentioned earlier, likelihood ratio tests are analogous to F-tests in a linear model, and indeed F-tests are a special kind of likelihood ratio test. So this function is designed to do the appropriate anova-esque hypothesis tests, regardless of what kind of model you give it. Also, why does the output say "Analysis of Deviance table"? To answer this, let's first go back to the output from summary().

Wald tests. You may have noticed that summary() returns, for each coefficient, a standard error, a z-value, and a p-value. We know that the coefficient estimate is the maximum likelihood estimate, but where does the standard error come from? The function assumes that the shape of the likelihood surface around the MLE is the shape of a normal distribution, in order to calculate an approximate standard error based on the curvature of the likelihood function near the MLE. This approximation becomes increasingly true as sample size increases. The z-value is then calculated in order to test whether the coefficient is different from zero, i.e. the z-value is $\frac{\theta_{MLE}}{se(\theta)}$, the coefficient estimate divided by its standard error. The z-value can then be used with the normal distribution to get a p-value for whether the coefficient is different from zero. This is called a Wald test, and it assumes that 1) the likelihood surface is shaped like a normal distribution, and 2) the sampling distribution for the MLE under the null hypothesis is also normal. So there's essentially a two step normal approximation going on here.

The Wald tests are fine for looking at a model quickly after fitting it, but if you actually want to test the significance of a coefficient you should use a likelihood ratio test. LRTs are more accurate because they rely on just one assumption (the LR is chi-square distributed) rather than two (the two step normal approximation). For linear models with normal error, summary() returns t-tests that are analogous to the Wald tests, but for the linear model case these tests are good to use because they don't require the large sample size approximations.

Deviance

Doing summary() on a poisson glm returns some info at the bottom, "null deviance" and "residual deviance". The deviance plays a role in generalized linear models that is analogous to the role of sums of squares in linear models, and it is used for residual diagnostics and certain kinds of significance tests.

When we defined the likelihood ratio test, the number we were interested in was

$$2 * \log\left(\frac{L(\theta_{MLE}|X)}{L(\theta_0|X)}\right)$$

or 2 times the log ratio of two likelihoods, the likelihood of the full model ($L(\theta_{MLE}|X)$) and the likelihood a restricted model with a term removed ($L(\theta_0|X)$). The deviance is defined similarly, except now we are comparing the 'full' model we are interested in to a 'saturated' model that has one parameter for each data point. What is the role of this saturated model? As I mentioned earlier, adding more parameters to a model will increase the likelihood, even if those parameters are just fitting noise rather than real effects. If each data point gets its own parameter, then that will yield a model with the highest possible likelihood for the data. The deviance is defined as

$$2 * \log\left(\frac{L(\theta_s|X)}{L(\theta_{MLE}|X)}\right)$$

Where $L(\theta_s|X)$ is the likelihood of the saturated model. So this number is comparing the likelihood of the model we fit (θ_{MLE}) to the highest possible likelihood ($L(\theta_s|X)$). This is a way to quantify how well the model fits the data, because it is quantifying how poorly the model fits relative to a 'perfect' model. summary() terms the deviance the 'residual deviance'. summary() also returns a 'null deviance', which is calculated in the same way, except now the saturated model is compared to a model that only has an intercept. So the null deviance basically tells you the upper limit on how poorly a model could fit the data, and the residual deviance tells you how poorly your model of choice fits the data.

Generalized linear models do not have a direct analogue to the R^2 of linear models, but there is a proliferation of pseudo-R^2 that attempt to give some sense of "on a scale from 0 to 1, how well does the model explain the data?". The deviances returned by glm can be used to calcuated one pseudo-R^2, called McFadden's:

$$1 - \frac{\text{residual deviance}}{\text{null deviance}}$$

or in R, 1 – model$deviance/model$null.deviance. If the full model is as good as the saturated model, the residual deviance will be 0 and the pseudo-R^2 will be 1. If the full model is no better than the null model, this quantity will be 0. For the example model of benthic richness on beaches, McFadden's pseudo-R^2 is 0.74, which is fairly high. This makes sense because there is large variation among beaches (and Beach is a term in the model), and because NAP is a pretty good predictor as well. Personally I never use or report pseudo-R^2, as I would rather visualize the model fit with plots, and because I am often interested in one term (e.g. tidal height) more than the whole model. Later we'll look at some ways to plot the effect of one term while controlling for variation explained by other terms.

When you do likelihood ratio tests with anova() or Anova(), it reports the results as 'Analysis of Deviance'. The reason is that likelihood ratio tests can be written in an

equivalent way as a comparison of deviances. I.e., the likelihood ratio test compares the different in log likelihood of two models, and this is equivalent to comparing the difference in the deviance of the two models. Some people like to talk about Analysis of Deviance because it is analogous to Analysis of Variance, and the Deviance is playing a role similar to the residual sums of squares in an anova. Personally I think it's clearer to just think about doing likelihood ratio tests.

Residuals

Another thing the deviance is used for is calculating residuals for a glm. In a linear model with normally distributed variation the residuals are

$$\varepsilon_i = Y_i - \mu_i$$

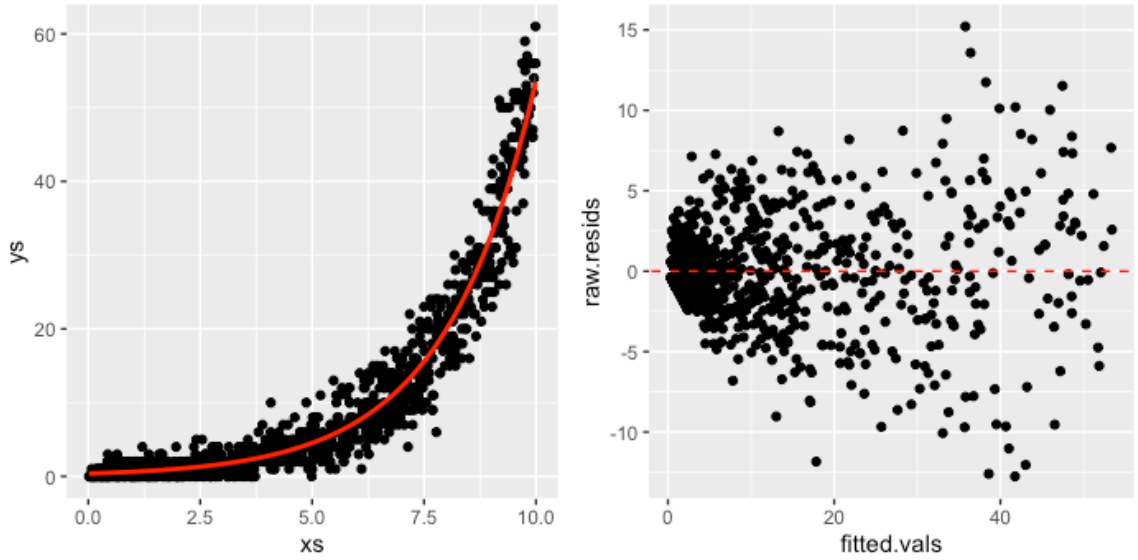
where residual ε_i is equal to the observed data Y_i minus the expected or predicted value μ_i. Then analysis of these residuals can be used to examine whether the model assumptions are met (linear relationships, constant residual variation across different values of the predictors, independence of residuals).

Residual diagnostics are a bit more difficult with GLMs, in part because the residuals are no longer supposed to be normally distributed, and because the variation in the residuals changes as the mean of the response variable changes. One way to make diagnostics more feasible is to "normalize" the residuals in some way.

For a Poisson regression the values $\varepsilon_i = Y_i - \mu_i$ can be calculated, and these are often called the "raw" residuals; in R these are called "response" residuals. However, these residuals are not as useful for diagnostics because of the *mean-variance scaling* of the Poisson distribution. We discussed this in lecture 2, let's review it now. The variance of the Poisson distribution is equal to the mean (usually called λ). This means that when the Poisson distribution is a good fit to the data, we expect that samples with a higher predicted value in the model (μ_i) will also have larger ε_i. In other words, samples predicted to have a high value of the response are also predicted to have high spread. We can visualize this by generating a fake regression where the data follow a Poisson distribution:

```
xs = runif(1000, min = 0, max = 10)
ys = rpois(1000,exp(-1 + 0.5*xs))
example.model = glm(ys ~ xs, family = poisson)
fake.data = data.frame(xs, ys, raw.resids = residuals(example.model,
type = 'response'), fitted.vals = fitted(example.model),
deviance.resids = residuals(example.model, type = 'deviance'),
pearson.resids = residuals(example.model, type = 'pearson'))

fake.data.plot = ggplot(fake.data, aes(xs, ys)) + geom_point() +
```

```
geom_smooth(method = glm, method.args = list(family = 'poisson'), col =
'red')
raw.resids.plot = ggplot(fake.data, aes(fitted.vals, raw.resids)) +
geom_point() + geom_abline(intercept = 0, slope = 0, col = 'red',
linetype = 2)
grid.arrange(fake.data.plot, raw.resids.plot, nrow = 1, respect = TRUE)
```



On the left is the fake data, with the glm fitted relationship. On the right is the raw residuals, extracted with residuals(example.model, type = "response"), plotted against the predicted values based on the regression, extracted with fitted(example.model). I.e. fitted() extracts for each data point the corresponding point on the fitted regression curve. The left plot is a standard plot for regression diagnostics, and in this case we can see how the spread in the residuals increases as the predicted value increases.

When we look at residual plots to assess the model assumptions, we are looking for a *lack of pattern*, so the fact that the residual plot for a Poisson model has a pattern by definition is a problem. The solution is to define a different kind of residual that accounts for the mean-variance scaling, and there are two common ways to do this. One is called the *deviance residual*. The deviance quantifies the lack of fit of the model, relative to a saturated (perfectly fit) model. The deviance is calculated using the log-probability of each data point under the full model and the saturated model. That means the deviance can be partitioned into the contribution made by each individual data point, similar to how sums of squares are partitioned in classic linear models. The deviance contributed by a single data point is called the *deviance residual*, and it quantifies how much that data point contributes to the model's overall lack of fit. So if the deviance residual is large, then that data point

is relatively far away from its predicted value. The calculation of the likelihoods and deviance automatically accounts for the mean-variance scaling of the Poisson distribution; so for a point to have a large deviance residual, then the point needs to be far from its predicted value, relative to what is expected from the mean-variance scaling.

The other common way to calculate residuals for a GLM is *pearson residuals*. These account for mean-variance scaling by just dividing the raw residual by the expected standard deviation for that sample, i.e.

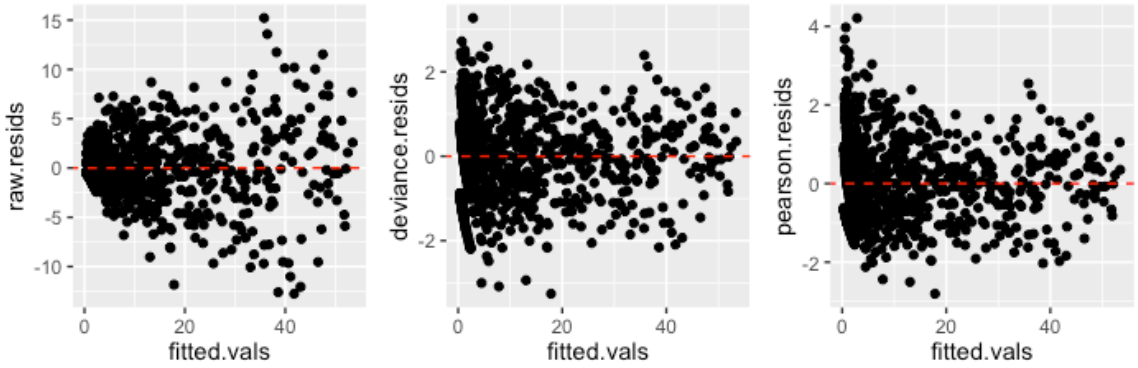
$$\varepsilon_i^P = \frac{Y_i - \mu_i}{\sqrt{\mathrm{Var}(\mu_i)}}$$

For the Poisson, we know that $\mathrm{Var}(\mu_i) = \mu_i$, i.e. the variance of the spread around a value will be directly proportional to the value. So for the Poisson

$$\varepsilon_i^P = \frac{Y_i - \mu_i}{\sqrt{\mu_i}}$$

Or in R, residuals(model, type = "pearson").

Now we can compare residual plots for the fake data:



On the left is residuals vs. predicted using raw residuals, in the middle is the deviance residuals, and on the right is Pearson residuals. Both the deviance and Pearson residuals get rid of the funnel shape in the plot, and give us the roughly patternless noise we're looking for. The two residuals often are very similar, but

sometimes they're not, which we'll probably encounter with binomial models. The residuals() function in R returns deviance residuals by default, while some of the function in the 'car' package use pearson by default.

Model diagnostics

After we fit a model, we want some way to assess whether the model makes sense. Plotting the fitted model against the raw data, as I did earlier for the RIKZ data, is always a good idea but is hard to do for more complex models, and doesn't reveal some problems as readily as residual plots.

We can also use the deviance or pearson residuals to look for large deviations from model assumptions. The most useful plots are probably 1) residuals vs. fitted values, and 2) residuals vs. individual predictors. Recall that the fitted values are the values predicted by the deterministic part of the model. So for a Poisson GLM we have a model of the form

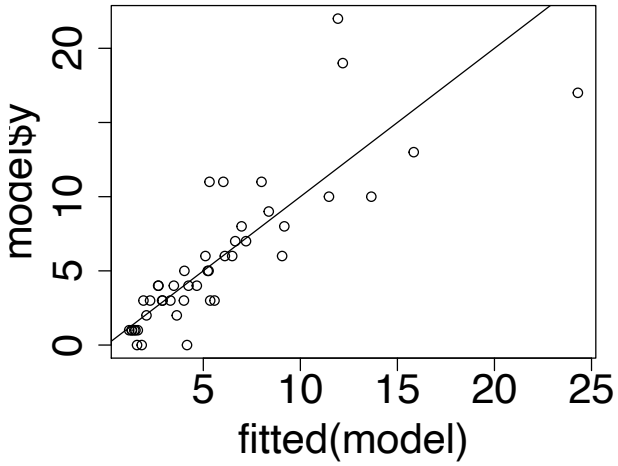
$$\mu_i = \exp(\beta_0 + \beta_1 * X_{1i} + \beta_2 * X_{2i} + \cdots)$$

$$Y_i \sim \text{Poisson}(\mu_i)$$

So for each sample Y_i in the dataset, the model coefficients will predict a particular value of μ_i. You can compare the fitted/predicted values to the real values with

```
plot(model$y ~ fitted(model))
```

Where I used model$y because for any model fit with lm() or glm(), the response variable is stored as model$y. So we can look at this plot

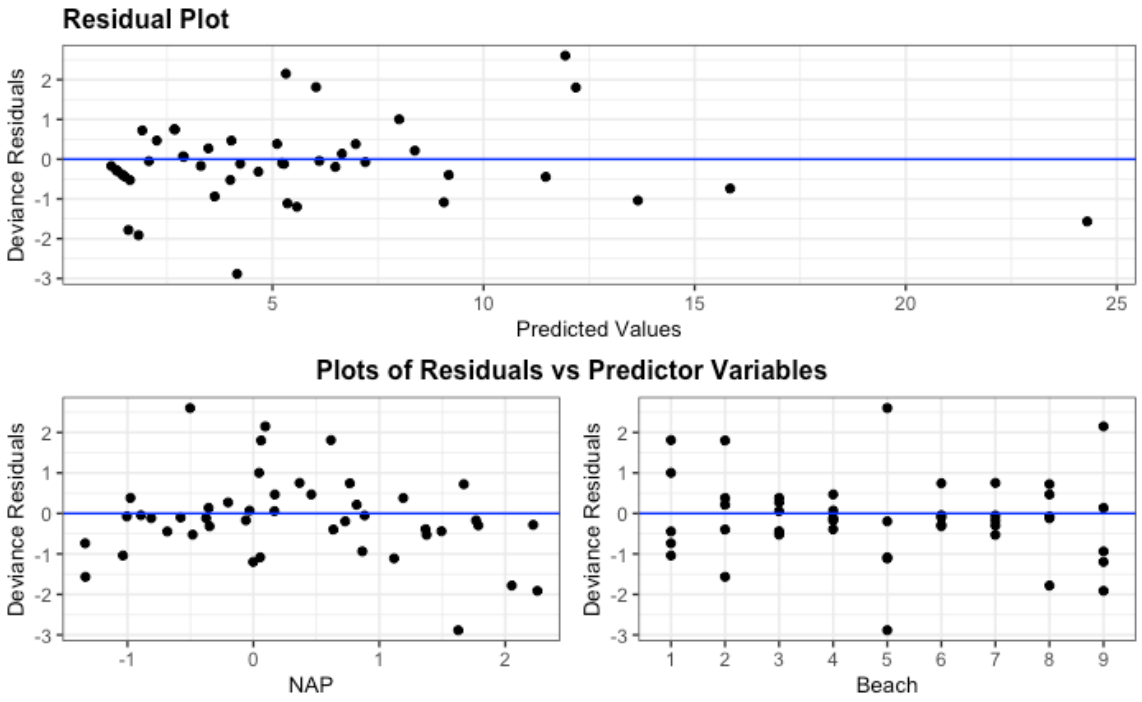


I've also plotted a one-to-one line with abline(a = 0, b = 1), because if the model is perfoming well then there should be a one-to-one relationship between the observed and predicted values. But it's usually better to plot the residuals, i.e. the deviation between the observed values and the predicted values, because 1) it's easier to detect an undesirable pattern, and 2) you can use some kind of normalized residuals for a GLM, as described above.

Residuals vs. fitted values will hopefully show a band of points with a mean of zero and no discernable pattern. If there is some kind of linear or nonlinear trend, the model is systematically under-estimating or over-estimating certain values of the response. This may indicate a poor choice of link function (because the link function relates the predictors to the response), or the need for some nonlinear term in the model. In addition, if there is a large change in spread of the residuals as a function of the fitted values, this may indicate problems with the assumed error distribution. This latter point is more challenging to evaluate, because the deviance and pearson residuals are approximately normally distributed under large sample size, but this might not apply to a particular analysis.

Plotting residuals vs. individual predictors has a similar purpose, but gives more information on whether a particular predictor is modeled well. E.g. a continuous predictor may exhibit evidence of nonlinearity, and a continuous or categorical factor may exhibit a lack of homoscedasticity.

Let's look at these plots for the RIKZ example, which are returned by functions in the ggResidpanel package:



These plots show the residuals vs. fitted values, deviance residuals vs. NAP, and deviance residuals vs. Beach, respectively. In this case these plots look OK, although it looks like the variance differs somewhat between different beaches. However, in this dataset there are only 5 samples per beach, and the beaches differ greatly in mean richness, which may tend to make the spread especially small when the mean richness is small, even after the mean-variance 'correction'. In addition, and residuals vs. NAP has a modest unimodal pattern, so this relationship may be captured more precisely with a nonparametric smoother (which we will cover in future lectures).

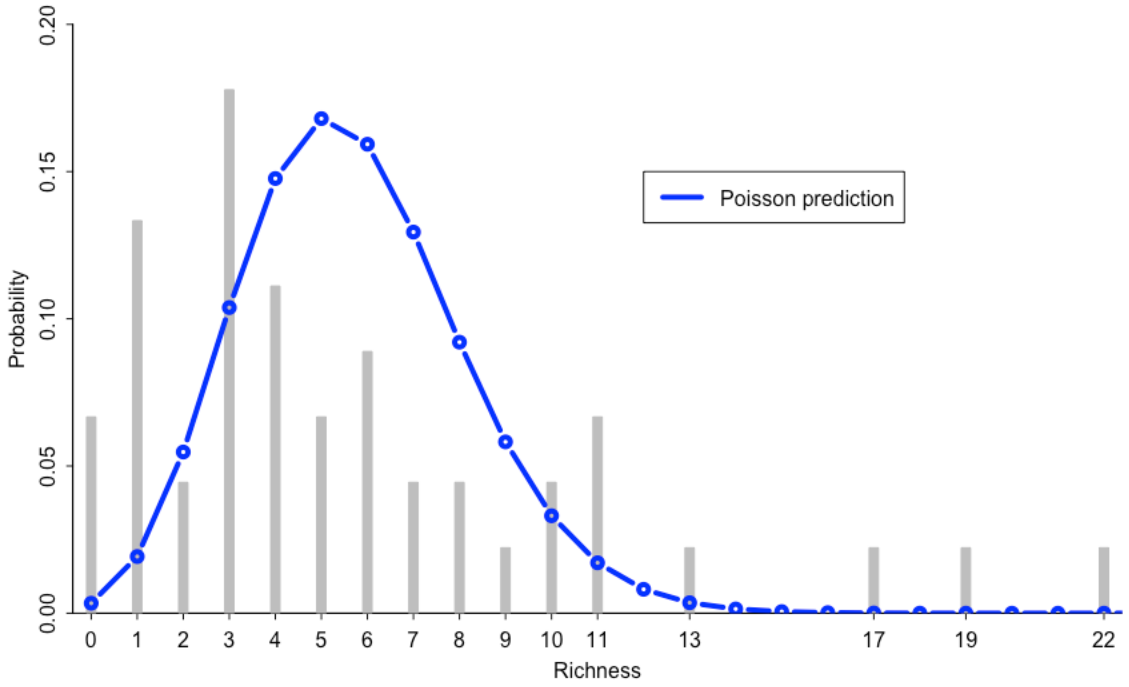
In general, it is most important to make some plots of the raw data and/or residuals to convince yourself that the deterministic structure of the model makes sense, and that the variability in the data does not show any especially weird patterns.

Overdispersion

The residual diagnostics just described are important for figuring out whether a GLM makes sense, but probably the most common and most important issue for GLM fits is *overdispersion*.

We've discussed several times that distributions like the Poisson and the binomial have a particular relationship between the mean and the variance, which I refer to as the mean-variance scaling. For the Poisson, the variance of the distribution is equal to the mean, and for the binomial the variance is a hump-shaped function of the mean, *n*p*(1-p)*. Probably the most common issue in applying these distributions to biological data is that biological data often show more variance than is expected based on mean-variance scaling. This too-much-variance is called overdispersion. For now we'll focus on the Poisson. Let's plot a histogram of species richness for all of the samples of the RIKZ dataset:

```
#histogram of RIKZ richness, and compare to the Poisson-predicted
densities
discrete.histogram(RIKZ$Richness, prob.col = 'grey', ylim = c(0, 0.2))
lines(0:25, dpois(0:25, lambda = mean(RIKZ$Richness)), col = 'blue',
lwd = 4, type = 'b')
legend(12, 0.15, lwd = 4, col = 'blue', lty = 1, legend = "Poisson
prediction")
```



I've also plotted the predicted frequencies from Poisson distribution that has the same mean. We can see that the two distributions don't match as well as they could; the real distribution has a lot more low values, too few intermediate values, and too many high values. What's going on here is that even though these two distributions have the same mean, the real data has greater variance than the Poisson. What causes the greater variance? Based on our prior analyses of this dataset, there are some obvious potential causes. We know that richness varies among beaches, and that richness varies with tidal height. The Poisson distribution is derived by assuming there is some underlying infrequent process that generates the data, and that this process happens at a constant rate. But if there are important sources of heterogeneity that are not accounted for, e.g. in environmental conditions, then this will lead to greater variability in the data than is expected.

Why is this important? Overdispersion is an important problem in GLMs because the variability in the data determines how confident we are in that parameter estimates. The Poisson (and the binomial) make a specific assumption about how much variability there is in the data, and this assumption is an implicit part of the likelihood calculations. If the data show greater residual variability than is assumed, then confidence intervals and p-values from likelihood ratio tests will be artificially small. The magnitude of this effect is directly proportional to how much excess variance there is.

How can overdispersion be quantified, so that we know whether it's a problem or not? Probably the best way is this formula:

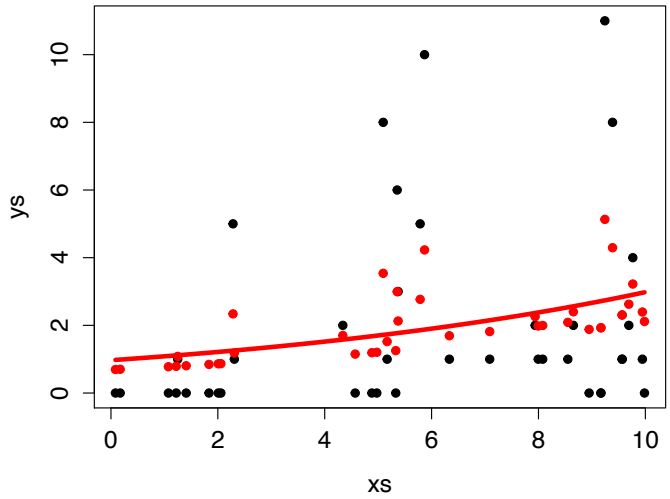
$$\varphi = \frac{\sum_i^n \varepsilon_i^2}{n - p}$$

Where φ is the *dispersion parameter*, ε_i is the *Pearson* residual for observation *i*, *n* is the number of observations in the dataset, and *p* is the number of parameters fit by the model. So this is the sum of the squared pearson residuals, divided by the model degrees of freedom. If there is no overdispersion then the dispersion parameter should be ~1, and a value greater than 1 indicates overdispersion. How big is too big? This is hard to say, and will depend on the situation, but in general the effect is proportional to the square root of the dispersion parameter. So if the φ is 4, then the confidence intervals for the model coefficients will be artificially small by a factor of 2. I've seen rule of thumb suggestions that above 1.5 is too high. In a perfect world R would print φ when you fit a GLM, but instead we can calculate it like this:

```
dispersion = function(model) {
  sum(residuals(model, type = "pearson")^2)/(length(model$y) -
length(model$coefficients))
}
```

You just have to plug in the name of your model, the other pieces are attributes of any glm fit. For the histogram of richness above, the dispersion parameter for a model with just an intercept (i.e. a model that just fits the mean of the distribution) is 4.08, which is consistent with our visual analysis.

Here's another example with some fake data:



If we observed the black dots and fit this relationship with a Poisson GLM, we would get the red line as our fitted relationship. The problem is that given the Poisson mean-variance scaling, the model is expecting that the variability around the fitted relationship looks more like the red points, which have a much narrower spread. In this case the dispersion parameter is about 4. When we do a likelihood ratio test on this fit, it says p = 0.0025. But if we correct for overdispersion (I will describe this shortly), then p = 0.14. So the primary issue with overdispersion is that it can substantially increase "false positives" when we are doing hypothesis tests.

How can we account for overdispersion? We will go through several alternatives, but probably the best one is to include predictors in your model that account for the extra variation. For example, in the earlier example with richness in the RIKZ dataset, we know that richness varies by beach as well as tidal height. Once we put these terms in the model (i.e., the model I fit last lecture), the dispersion parameter goes from ~4 to 1.2. So in this case, if we account for variation in the response due to important environmental predictors, the overdispersion goes away.

Quasi-Poisson

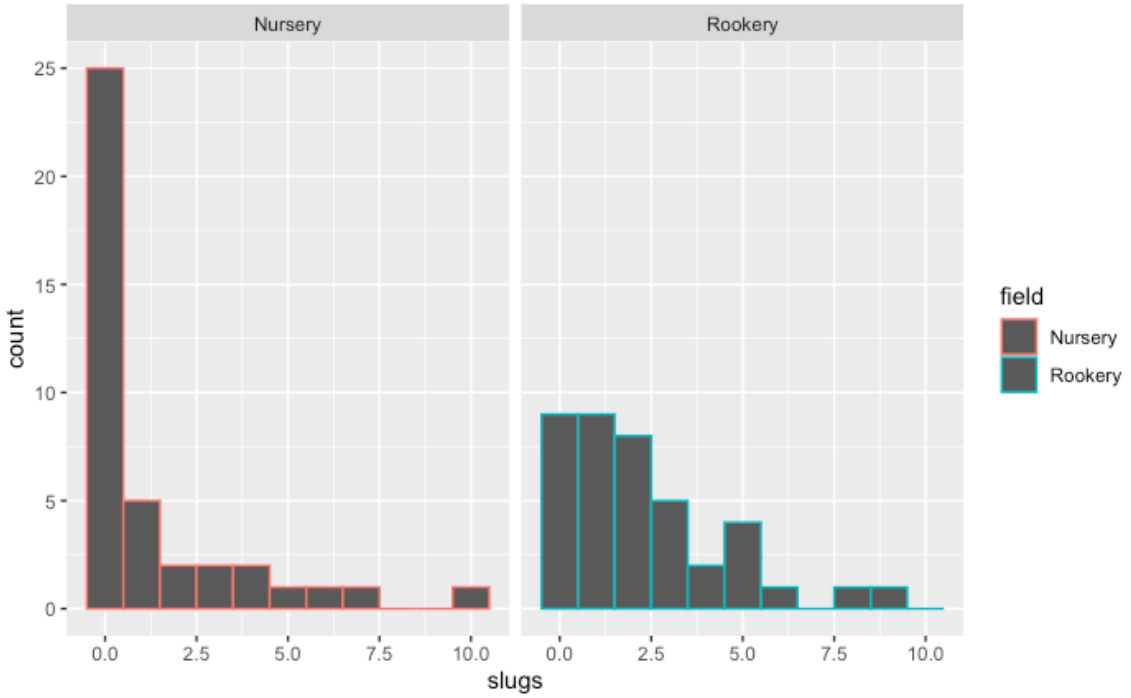
If you've already constructed your model and don't have any obvious missing terms to account for extra variation, what do you do if there is still overdispersion? A simple solution is to use the so-called quasi-Poisson option with the glm() function. For the Poisson distribution the variance is equal to the mean, so for a particular fitted value μ_i, the expected spread around that value has a variance of μ_i. The quasi-Poisson approach says "We don't know exactly what distribution the data has, but we'll assume that the variance is *proportional* to the mean, i.e. $V(\mu_i) = \varphi\mu_i$", where φ is the dispersion parameter I defined earlier. So this approach

assumes that the distribution is Poisson-like in terms of how the variance changes with the mean, but it allows the variance to be bigger (or smaller) by a factor of φ. This approach is kind of weird because the model-fitting algorithm is similar to what is used for the Poisson, but the model doesn't actually have a real probability distribution, and therefore doesn't have a likelihood, which is why this kind of approach is called quasi-likelihood.

The dispersion parameter is estimated as part of fitting the model, and then used to correct any standard errors, confidence intervals, or hypothesis tests. Specifically, the standard errors and confidence intervals from the Poisson model get multiplied by sqrt(φ) to get corrected intervals. Likewise, the likelihood ratio statistic is corrected with φ to get a quantity that has an approximate F-distribution. Theory shows that the overdispersion doesn't affect the parameter estimates *per se*, only the uncertainty around those estimates, and so the quasi-Poisson is a simple and useful approach for count data with excess variance.

Here's an example of using the quasi-Poisson. 40 tiles were places in each of two fields in England, as part of a pilot study to see if slug abundance differed between the fields. For this crucial scientific question we will use the most appropriate and sophisticated statistical methods. At some point the number of slugs under each tile was counted, so we have 40 counts from each site. Since there's only two sites, we can go ahead and look at a histogram of the slug counts at the two sites:

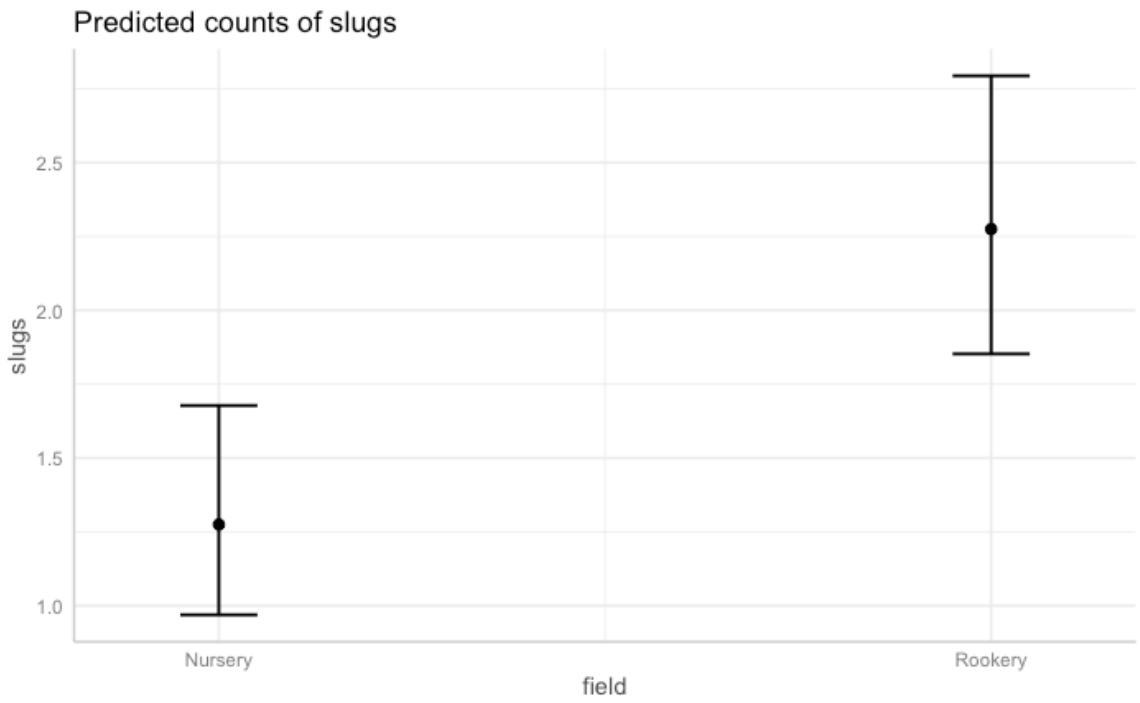
```
#slugs overdispersion example
slugs <- read_table('http://www.bio.ic.ac.uk/research/mjcraw/statcomp/data/slugsurvey.txt')

ggplot(slugs, aes(slugs, col = field)) + geom_histogram(bins = 11) + facet_wrap(~field)
```



It looks like there might be more slugs at the Rookery site than the Nursery site, or maybe not. In addition the distributions look pretty skewed, even for Poisson distribution with a mean close to zero. So to compare the sites I'll first fit a Poisson, and look at the dispersion parameter.

```
#poisson model
model.poisson = glm(slugs ~ field, data = slugs, family = poisson)
plot(ggeffect(model.poisson))

Anova(model.poisson)
```

```
## Analysis of Deviance Table (Type II tests)
##
## Response: slugs
##       LR Chisq Df Pr(>Chisq)
## field     11.4  1    0.00073 ***
## ---
## Signif. codes:  0 '***'
```



The fitted means show about twice as many slugs at the Rookery site, and the likelihood ratio test is highly significant. But when we consider overdispersion, it looks potentially problematic:

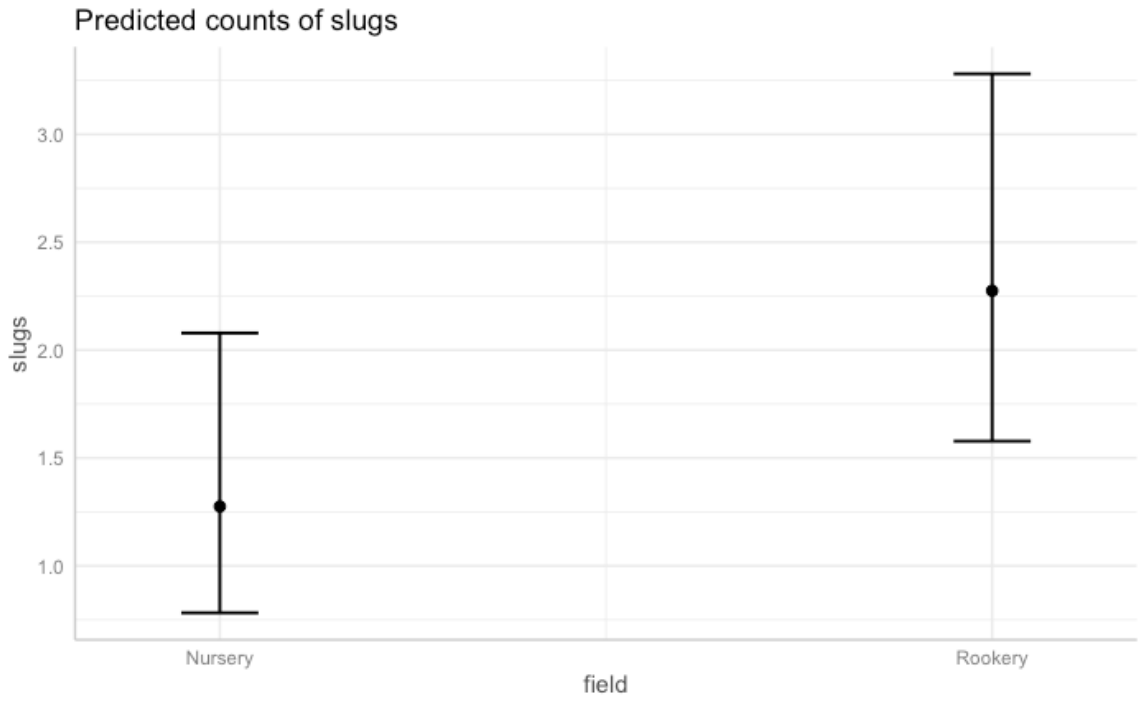
```
#dispersion parameter function
overdis = function(model) {
  sum(residuals(model, type = "pearson")^2)/(length(model$y) - length(m
odel$coefficients))
}
overdis(model.poisson)

## [1] 3.173
```

The dispersion parameter is 3.2. I'll fit a quasi-Poisson model and compare the results:

```
#quasipoisson model
model.quasipoisson = glm(slugs ~ field, data = slugs, family = quasipoi
sson)

summary(model.quasipoisson)

##
## Call:
## glm(formula = slugs ~ field, family = quasipoisson, data = slugs)
##
## Deviance Residuals:
##    Min      1Q  Median      3Q     Max
```

```
## -2.133  -1.597  -0.952   0.458   4.873
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept)     0.243      0.249    0.97    0.333
## fieldRookery    0.579      0.312    1.86    0.067 .
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for quasipoisson family taken to be 3.173)
##
##     Null deviance: 224.86  on 79  degrees of freedom
## Residual deviance: 213.44  on 78  degrees of freedom
## AIC: NA
##
## Number of Fisher Scoring iterations: 6
```

```
plot(ggeffect(model.quasipoisson))
```



The fitted means are the same, because the model parameters don't change with the quasi-likelihood method, only the uncertainty, and indeed the error bars on the fitted means are larger. Note that summary() returns the estimated dispersion parameter, which is the same as the one I calculated with my function. So in practice, if you don't want calculate the dispersion parameter yourself, you can just fit a quasipoisson model and see how big the parameter is. To test the difference between means we can use Anova() with test = 'F', which uses a corrected likelihood ratio statistic and compares that to an F distribution.

```
Anova(model.quasipoisson, test = 'F')
```

```
## Analysis of Deviance Table (Type II tests)
##
## Response: slugs
##              SS Df   F Pr(>F)
## field      11.4  1 3.6  0.061 .
## Residuals 247.5 78
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

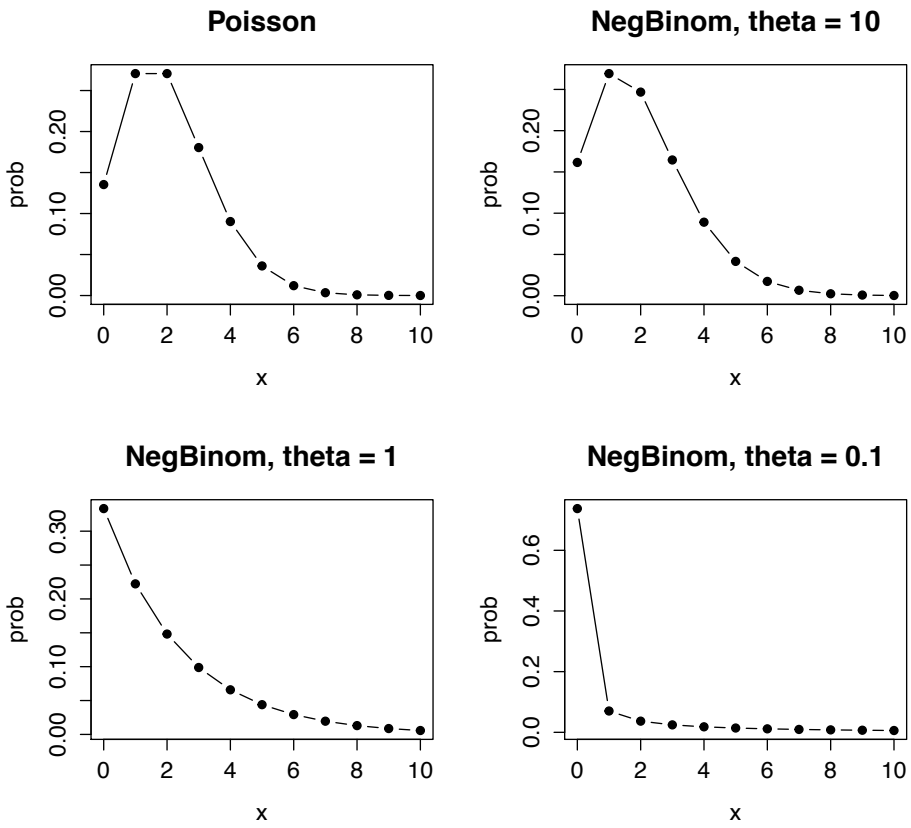
After correcting for overdispersion, the effect of field is not significant at a 0.05 cutoff. So do the two fields have different amounts of slugs or not? In some sense this is a trivial question, because if we had a large enough sample size we would probably be able to detect some difference. Nonetheless, many questions in biology come down to 'are these two things different?', especially if we are doing a controlled experiments, as opposed to the observational example. And this example shows that accounting for overdispersion is important, because you can easily detect a lot of spurious effects if you're using a Poisson model with highly overdispersed data.

The quasi-likelihood approach is convenient, but it has one major downside, which is that we are fitting a model that doesn't have a real probability distribution. The model is fit using the assumption that the variability is Poisson-like, but with an extra dispersion term. As a result, the fitted model has no likelihood. We can still do the F-test described above, which is often sufficient, but if we want to compare models with AIC (we'll get to this later), a likelihood is necessary.

Negative Binomial

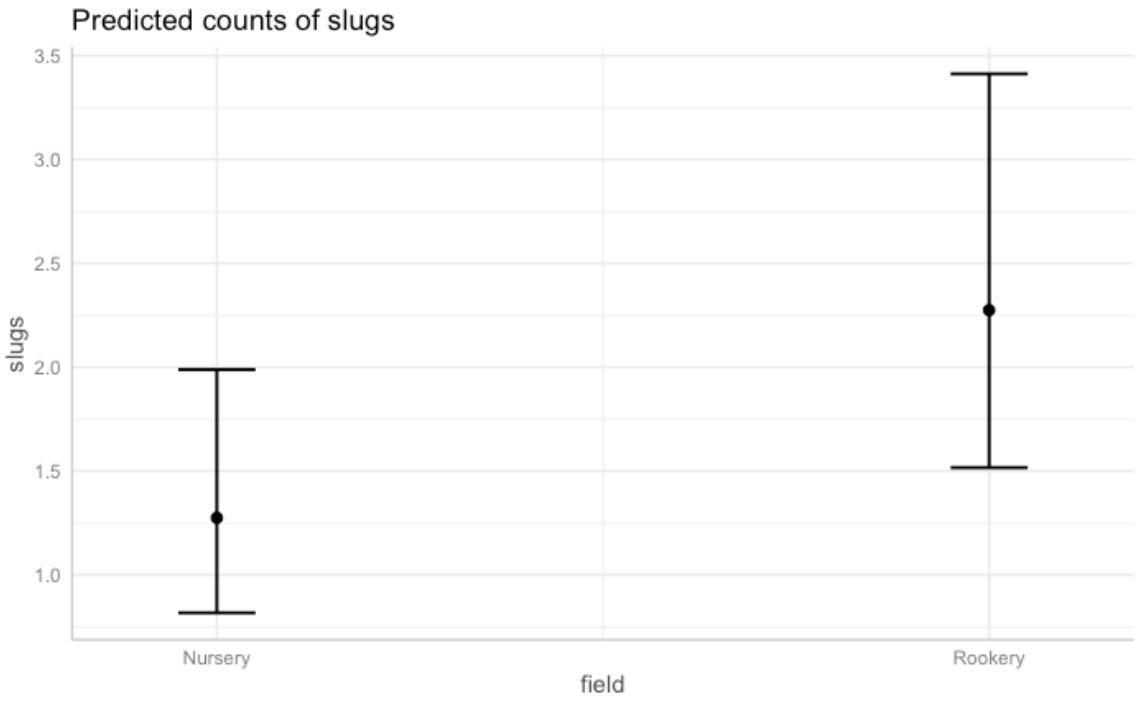
There are two other ways to deal with overdispersion in the Poisson, 1) use negative binomial distribution instead, and 2) use random effects to account for the extra variability. The second solution is my preferred one, because I'm often using a mixed model anyways, but we will get to mixed models later.

In the second lecture I mentioned the negative binomial distribution. It has a similar shape to the Poisson, but has an extra parameter that allow it to have a higher variance. This parameter has a confusing variety of names. It is often called the dispersion parameter (but don't confuse it with the dispersion parameter φ defined above), and the R function glm.nb calls it theta. Here's an example of what it looks like to vary theta:



When this parameter is large, the distribution looks like the Poisson, and as it gets small the distribution gets increasingly spread out and pushed up against zero. The shapes with smaller theta are similar to what the raw data looks like for the slug example. Let's fit a GLM with a negative binomial distribution to this data. We have to use a special function, glm.nb(), in the MASS package. But the syntax is the same.

```
library(MASS)
model.negbin = glm.nb(slugs ~ field, data = slugs)
plot(ggeffect(model.negbin))
```



The plot looks very similar to the quasipoisson fit. We can do a likelihood ratio test on this model:

```
Anova(model.negbin)
```

```
## Analysis of Deviance Table (Type II tests)
##
## Response: slugs
##       LR Chisq Df Pr(>Chisq)
## field     3.56  1      0.059 .
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

And the p-value is very similar to the quasipoisson as well. It is possible, though maybe not likely, that the data could be overdispersed even compared to the negative binomial. We can check with our overdispersion function:

```
overdis(model.negbin)
```

```
## [1] 1.075
```

Looks like the negative binomial model successfully accounts for the overdispersion.

When should you use the quasi-Poisson vs negative binomial approaches? In most cases it probably doesn't matter and they will give similar results. If you want a normal likelihood model that lets you use model selection techniques like AIC, the negative binomial is the way to go.