7

Getting Started with Generalized Linear Models

7.1 Introduction

The previous two chapters have focused largely on the linear model, that class of model that includes our favourites, including regression, ANOVA, and ANCOVA. The response variables in these tools share some important features in common: we assume they are continuous variables that can take both positive and negative values and can be fractions. We also assume these data are unbounded, though in practice they may not be (e.g. body size can't be negative). These models all also assume normally distributed residuals. And they also assume something special: a constant mean–variance relationship (remember that panel in the diagnostics?).

We always checked these assumptions and found them not badly violated, so it was sensible to use general linear model. Often, however, we work with response variables for which the normality assumption is violated. We often know this in advance, because in the biological sciences we frequently collect data that are bounded and have other features that make this assumption fail. Sometimes we collect data that are integer-valued and bounded, that can not be fractions or negative, like the number of babies.

Getting Started with R Second Edition. Andrew Beckerman, Dylan Childs, & Owen Petchey: Oxford University Press (2017). © Andrew Beckerman, Dylan Childs, & Owen Petchey.
DOI 10.1093/oso/9780198787839.001.0001

Downloaded from https://academic.oup.com/book/27290/chapter/196938074 by University of Glasgow user on 25 September 2024

Sometimes we collect data that are strictly binary (e.g. presence/absence data). These types of data can violate the assumptions of the general linear model.

Historically, for these types of data, we often invoked *transformations*. We may have used a `log10` transformation for counts, or an `arcsin(sqrt())` transformation for proportions. But with the advent of computing speed on our desktop, and developments in statistics, we can now just use a powerful and effective tool: the generalized linear model (GLM). However, this has not stopped people from discussing transformations at length. Just search the journal *Methods in Ecology and Evolution* for the three papers since 2010, or look on their Facebook (yes) page for the 'should you log-transform counts' saga (spoiler—sometimes it is OK).

7.1.1 COUNTS AND PROPORTIONS AND THE GLM

Before we dive into using a GLM, let's reveal a bit more about biological data that requires the GLM tool. Count data come in many forms. Biological questions often involve counting a number of things in a unit of time or space, such as the number of individuals in a population, species in a patch, or parasites present within an individual. We usually want to relate these counts to other variables. For example, we might be interested in how the number of offspring produced by a female over her lifetime is related to body mass or age. These are questions where the response variable is counts and the goal is to understand how the rate of occurrence of events (e.g. births) depends on the other variables (e.g. body size). Count data are bounded between zero and infinity, violate the normality assumption, and don't have a constant mean–variance relationship.

Data relating to proportions are also common. A common type of data captures whether or not an event happens, such as does an animal die, does a plant flower, or is a species present in a grid cell? Or we may collect data on sex ratios. Once again, we usually need to relate these occurrences to one or more explanatory variables. For example, we might be interested in whether death is related to the concentration of pesticide an insect is

Downloaded from https://academic.oup.com/book/27290/chapter/196938074 by University of Glasgow user on 25 September 2024

exposed to. This is a question where the response variable is either *binary* (i.e. an indicator variable of dead or alive) or another kind of *count*. In the case of survival, we can work either with an indicator variable that describes whether each individual survives (binary, 0 or 1), or with a summary count that describes how many individuals in a group have died. In both cases the goal is the same: we want to understand how the *probability* of an event occurring depends on the explanatory variables. These data are called binomial. They are also bounded, violate the normality assumption, and don't have a constant mean–variance relationship.

We think you probably get it . . . boundedness and non-constant mean–variance relationships lead to the *Generalized Linear Model*. Here we introduce the GLM as a solution to these problems, an alternative statistical modelling framework that properly accounts for the properties of such variables.

7.1.2 KEY TERMS FOR GLM MODELS

In the following sections, we show the basics of how to work with GLMs in R. This is not an easy task, as the theory and implementation of a GLM are quite different from a linear model. However, we aim to explain, in practical terms and in our 'getting started' voice, some of the nuances, details, and terminology associated with GLMs.

We start by introducing three important terms and their meaning. We will use these in the subsequent example, and want to provide a semi-formal definition in advance. And one of them is the 'family'. Who can argue with learning about family?

1. *Family* This is the probability distribution that is assumed to describe the response variable (also referred to as the *error structure*). By the way, a probability distribution is just a mathematical statement of how likely different events are. The Poisson and binomial distributions are examples of families.
2. *Linear predictor* Just as in a linear model, there is a linear predictor, i.e. an equation that describes how the different predictor variables

Downloaded from https://academic.oup.com/book/27290/chapter/196938074 by University of Glasgow user on 25 September 2024

(explanatory variables) affect the expected value of the response variable (e.g. birth rates or probabilities of death in the discussion above).

3. *Link function* This one can be tricky at first glance, but, essentially, the clue is in the name. The link function describes the mathematical relationship between the expected value of the response variable and the linear predictor—it 'links' these two parts of a GLM.

Don't worry if these definitions aren't totally clear at the moment. Return to them after you've been through this chapter, and they should (will, we promise) make more sense.

We're only going to look at one type of GLM, the one that often is a good starting point when you have count data. We will also always say 'general linear model' to refer to models fitted with **lm**() and 'generalized linear model' for those fitted with **glm**(). Finally, there is a lot more 'theory' in this chapter because experience has shown us that even when people have been taught GLMs, they miss important details like the significance of link functions. . . we think you'll appreciate this bit of theory, and it also helps you understand better the theory for the general linear models covered in the previous chapters.

7.2 Counts and rates—Poisson GLMs

7.2.1 COUNTING SHEEP—THE DATA AND QUESTION

We start by counting sheep—don't go to sleep. This is an example where the response variable is a count variable like we discussed above. Key to understanding the question these data were collected to answer is to remember that, with such counts, our the goal is to understand how the *rate* of occurrence of events (counts of babies produced) depends on one or more explanatory variables. That might seem a little cryptic at the moment, but trust us, it will make sense by the end of this section.

Our example can be thought of as a study of natural selection. Hirta is a small island off the west coast of Scotland that is home to an unmanaged

Downloaded from https://academic.oup.com/book/27290/chapter/196938074 by University of Glasgow user on 25 September 2024

population of feral Soay sheep (unmanaged *and* feral; yikes!). These sheep have been the subject of a great deal of ecological and evolutionary research, including studies of factors related to female (i.e. ewe) fitness. One way to measure fitness is to count the total number of offspring born to a female during her life, called 'lifetime reproductive success'. So, our response variable is counts of offspring. Lifetime reproductive success data are a good example of counts that are often poorly approximated by a normal distribution. It seldom appropriate to model them with a general linear model.

Let's assume we have measured the lifetime reproductive success of some ewes in the study population, alongside a standardized measure of average body mass (kg). The question we focus you on is whether lifetime reproductive success increases with ewe body mass; do bigger mums make more babies? If it does, and there are heritable differences in mass (offspring are like their parents), then in the absence of any biological constraints (a very strong assumption!) we should expect selection to increase body mass over time. Our goal here is to evaluate the hypothesis that body mass and fitness are positively associated in Soay sheep.

The data for this example are stored in a file called `soaysheepfitness.csv` (get it the same way you did for all other datasets). First we need to read this data into R and check its structure. Make sure you set this script up exactly the same way you did before, clearing the decks and getting ***ggplot2*** and ***dplyr*** loaded and ready to use. Then:

```
soay <- read.csv("soaysheepfitness.csv")
glimpse(soay)
```

The data are very simple—there are 50 observations and only two variables. The body.size variable is the standardized measure of average body mass (in kg) of each ewe, and fitness is her lifetime reproductive success.

As always, we should make a graph to summarize the key question. A scatterplot of fitness against body mass is the obvious choice (Figure 7.1). In addition to plotting the raw data, we can also display what a linear model might look like fitted to the data, and a non-linear relationship

Downloaded from https://academic.oup.com/book/27290/chapter/196938074 by University of Glasgow user on 25 September 2024

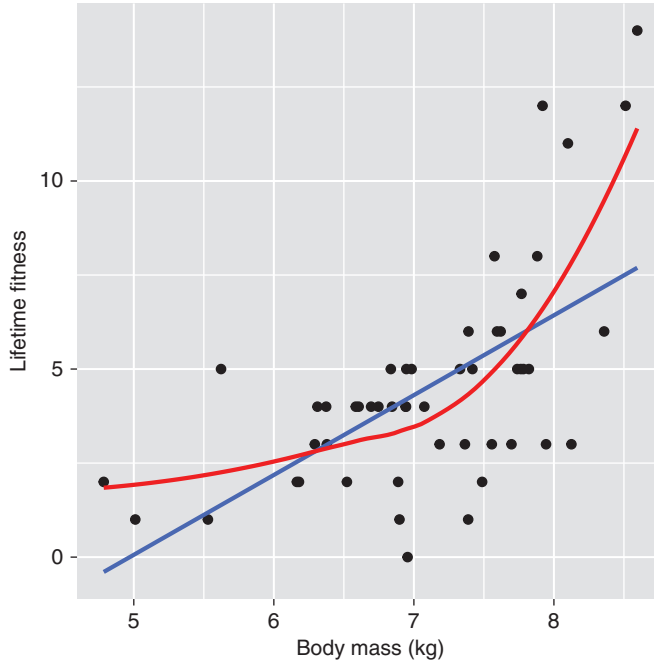

Figure 7.1 Fitness (number of offspring) versus mother's weight.

too. We can actually do both of these super-quickly using **geom_smooth**() from ***ggplot2***:

```
ggplot(soay, aes(x = body.size, y = fitness)) +
  geom_point() +
  geom_smooth(method = "lm", se = FALSE) +
  geom_smooth(span = 1, colour = "red", se = FALSE) +
  xlab("Body mass (kg)") + ylab("Lifetime fitness")
```

The blue line produced by **geom_smooth**(method = "lm", se = FALSE) shows a fitted linear regression line (remember using this in Chapter 4?), while the red line generated by **geom_smooth**(span = 1, colour = "red", se = FALSE) shows the non-linearity using a more flexible statistical model. The curve is produced by something called a local regression—we're not going to worry about how this works here other than to point out that the

Downloaded from https://academic.oup.com/book/27290/chapter/196938074 by University of Glasgow user on 25 September 2024

`span = 1` bit controls how 'wiggly' the line is (change it to 0.5 if you don't believe us).

Both lines clearly indicate a strong positive relationship between fitness and body size, whereby larger ewes have more offspring over their lifetime. This probably isn't very surprising. If it is, back to school. Big mums simply have more resources to 'spend' on reproduction. But it is also fairly obvious that the straight-line relationship (blue line) is rubbish and doesn't capture the general pattern very well. There is a degree of upward curvature in the fitness–size relationship, captured by the wiggly red line.

Is this a problem? Probably. We might deal with this non-linearity by using some kind of transformation or including a squared term in our regression model. However, there are other, more subtle problems in these data. To truly understand these problems, and the ultimate value of the generalized linear model, we are going to first carry out the 'wrong' analysis using the familiar linear regression model. We'll then use our favourite diagnostic plots to identify these problems. This is a much better strategy than trying to identify them by endlessly staring at the raw data. Once we understand the data and their problems, we'll carry out the 'right' analysis using generalized linear model. This approach will help you understand why GLMs are useful, and on the way you'll learn a little bit about how they work.

7.3 Doing it wrong

You know how to fit a general linear model with the **lm**() function and generate diagnostic plots (Figure 7.2) with the **autoplot**() function from ***ggfortify*** . . .

We hope you did make the model. And these look terrible, don't they? Finally, terrible diagnostics. But let's remember that we have just fitted a straight-line relationship to data that look non-linear and assumed normally distributed residuals with constant variance. The diagnostic plots are telling us there are several problems with these assumptions. In fact, they are telling us that we have violated nearly all of them.

Downloaded from https://academic.oup.com/book/27290/chapter/196938074 by University of Glasgow user on 25 September 2024

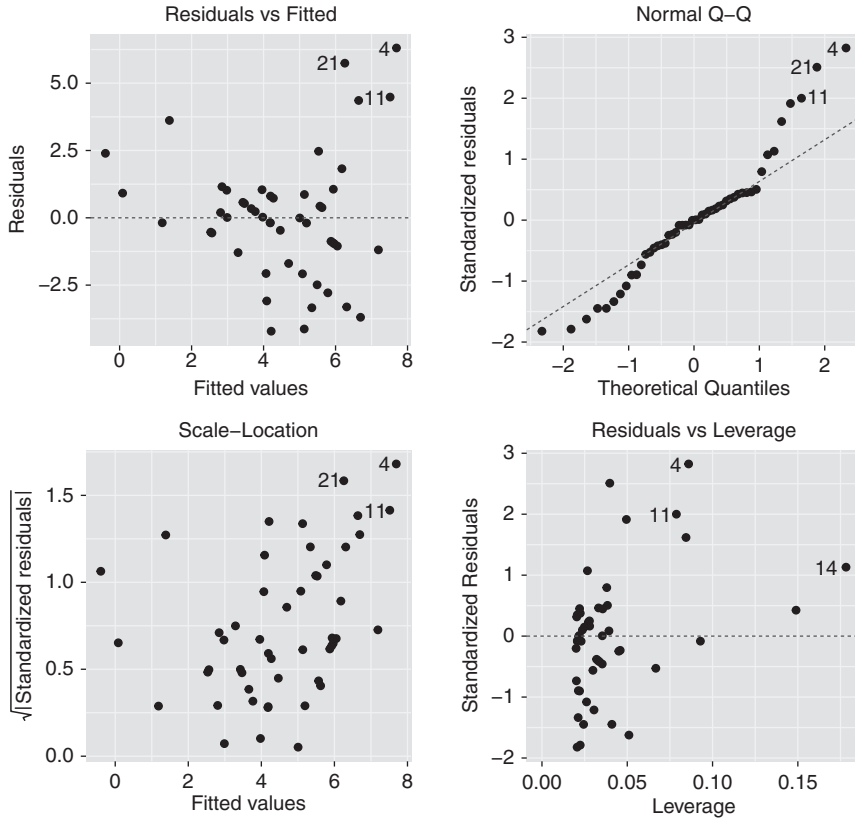

Figure 7.2 Diagnostic graphs for linear model of fitness – body mass relationship.

7.3.1 DOING IT WRONG: DIAGNOSING THE PROBLEMS

The plot of residuals vs fitted values (upper left panel) tells us that the systematic part of the model is not very good, i.e. there is a clear pattern in the relationship. Actually, we can say a bit more than this. The U-shape indicates that a straight-line model fails to account for curvature in the relationship between our two variables. We underestimate fitness at small body sizes, then overestimate it at medium sizes, and then underestimate it at large sizes.

The normal Q–Q plot (upper right panel) is also a problem. Most of the points should lie on the dashed line. Instead, points corresponding to

Downloaded from https://academic.oup.com/book/27290/chapter/196938074 by University of Glasgow user on 25 September 2024

the most positive 'theoretical quantiles' are consistently above the line and those associated with the most negative values are below it. It looks like our normality assumption is way off. Again, we can say more if we know what we're looking for. This particular pattern occurs because the distribution of the residuals is not symmetric; it is skewed to the right. If you want, make a histogram, go ahead . . .

The scale–location plot (bottom left panel) shows a positive relationship between the absolute size of the residuals and the fitted values. This reflects the way the fitness values are scattered around the red line in Figure 7.1. There is more vertical spread in the data at larger values of fitness. We can say there is a positive mean–variance relationship: large predicted fitness values are associated with more variation in the residuals. This is typical of count data such as these.

At least the residuals–leverage plot (bottom right panel) isn't too bad! There are no really extreme standardized residual values, so we don't seem to have any obvious outliers, and none of the observations are having too much of an effect on the model.

Overall, this exercise in fitting what we know to be the wrong model, the normal linear regression model, is not doing a very good job of describing these data. We're going to fix the model, but first we need to learn a little bit about a new distribution.

7.3.2 THE POISSON DISTRIBUTION—A SOLUTION

The problem with our linear regression model is that the normality assumption just isn't appropriate for raw, untransformed count data. Why are count data unlikely to be normally distributed? If we think about the properties of the normal distribution, we can list a few fairly obvious reasons:

1. The normal distribution concerns continuous variables (i.e. those that can take fractional values). However, count data are discrete. It is possible for a ewe to produce 0, 1, 2, or 3 lambs, but impossible for her to give birth to 2.5 offspring.

Downloaded from https://academic.oup.com/book/27290/chapter/196938074 by University of Glasgow user on 25 September 2024

2. The normal distribution allows negative values, but count data have to be positive (we'll include 0 here). A ewe may have 0 offspring, but she certainly cannot have –2 offspring. Isn't biology cool?
3. The normal distribution is symmetrical, but counts are often (but not always) distributed asymmetrically, in part because they cannot be negative. This might not be obvious to you, but it's true.

The normal distribution is just not a very good model for many kinds of count data. On the other hand, the Poisson distribution *is* a good starting point for analysis of *certain* kinds of count data. This isn't a statistics book, so we aren't going to explain the properties of the Poisson distribution in detail. A visual description is useful, though. Figure 7.3 shows three Poisson distributions, each with a different mean. The *x*-axis shows a range of different possible values (counts), and the *y*-axis shows the probability of each value.

We can see why the Poisson distribution is such a good candidate for count data, and for the problems we identified above, when fitting the linear regression model:

- Only discrete counts (0, 1, 2, 3, . . .) are possible. These are bounded at 0, and while very large counts are possible, the probability of one occurring is very low.

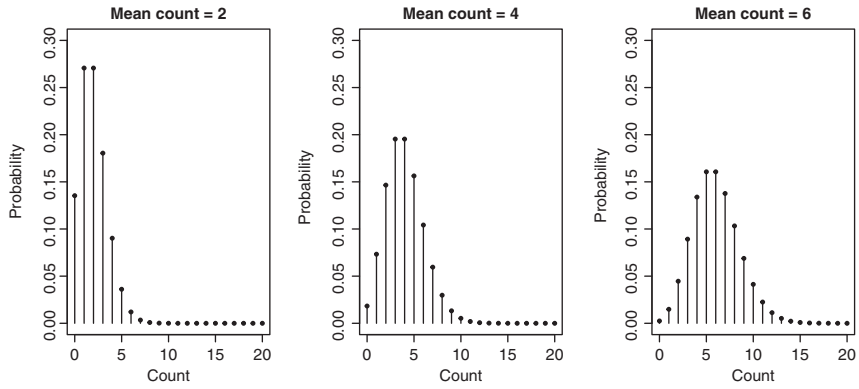

Figure 7.3 Examples of the Poisson distribution

Downloaded from https://academic.oup.com/book/27290/chapter/196938074 by University of Glasgow user on 25 September 2024

- The variance of the distribution increases as the mean of the distribution is increased. Visually, this corresponds to a widening of the base of the distribution as the mean grows.

The Poisson distribution is best suited for analysing 'unbounded' count data. *Huh? What do you mean by 'unbounded'? You just said the data were bounded!!!* Yes, we did. But this just refers to the fact that there is no upper limit to the values the count variable might take. For example, a female Soay sheep could produce 2, 5, or even 10 offspring over her lifetime. In reality there must be some biological constraints on lifetime reproductive fitness—a ewe will never produce 100 offspring—but that limit isn't something we know. The unbounded count assumption is just a useful approximation to reality.

So how do we model the relationship between the fitness of Soay ewes and their body mass using a Poisson distribution? By using a generalized linear model, of course! However, before we can use a GLM, we have to understand its component parts. Sorry, but we need a tiny bit more theory.

7.4 Doing it right—the Poisson GLM

7.4.1 ANATOMY OF A GLM

We mentioned three important terms related to the GLM at the beginning of this chapter: the 'family', the 'linear predictor', and the 'link function'. You have to understand at least roughly what these mean to use GLMs without making a terrible mistake. We're just going to give you a quick tour so that we can finally get up and running with GLMs in R.

The family

The family (or 'error') part of a GLM is not too difficult to understand. This bit just determines what kind of distribution is used to describe the response variable. A *general* linear model always assumes normality—that's pretty much its defining feature in fact—but with a *generalized* linear

Downloaded from https://academic.oup.com/book/27290/chapter/196938074 by University of Glasgow user on 25 September 2024

model there are several different options available. We can use a Poisson distribution (no surprises there), a binomial distribution (more on that later), or a gamma distribution (for positive-valued, continuous variables), plus a few other more exotic versions. Each of these is appropriate for specific types of data, but taken together, these different GLM families allow us to work with a wide range of response variables. GLMs supercharge your data analysis.

The linear predictor

Although you might not realize it, you already know about the GLM linear predictor. Every time you build a model using **lm**() you have to supply it with at least some data and a special little R formula to define the model. The job of that formula is actually to define the linear predictor (remember, general linear models have these too).

Let's try to understand that linear predictor. It's most easily understood with simple regression. We can use the Soay example for this. We fitted the bad regression using **lm**(fitness ~ body.size, . . .). That tells R to 'build me a model for the predicted fitness, with an intercept and a body size slope term.' It looks like this:

$$\text{Predicted Fitness} = \text{Intercept} + \text{Slope} \times \text{Body Size}.$$

We don't have to tell R to include the intercept; it adds it automatically because models without intercepts are seldom sensible. More, wait, most importantly, the bit of that little equation on the right side of the = is the linear predictor! So the linear predictor is really just 'the model', and all those coefficients shown by **summary**() are just estimates of different intercepts and slopes in the linear predictor. We really meant it when we said you already knew about the linear predictor.

By the way, it's called a linear predictor because we 'add up' the component parts, and every part is either an intercept or a slope term. Every single model in the previous couple of chapters—regression, ANOVA, ANCOVA—can be written down as a linear predictor, although models including factors are a little more tricky (so we won't try to do it).

Downloaded from https://academic.oup.com/book/27290/chapter/196938074 by University of Glasgow user on 25 September 2024

The link function

So that's the linear predictor sorted. What about the link function? This is the one part of GLMs that tends to confuse people. It confuses people a lot. But start simple. We have some problems, Houston, with our diagnostic plots. They are caused by bounded, integer data where the variance increases with the mean, in this case. The combination of choosing the right family and a link function with it can solve these problems.

A good way to start thinking about the link function is to demonstrate why we need it. Think about the Soay sheep fitness linear regression model. Let's plug some numbers in. What happens if the estimated intercept is –2 and the slope is +1.2? Let's try to predict the average number of offspring produced by a 2 kg ewe: $-2 + 1.2 \times 2 = 0.4$. We're only predicting the average, not an actual number, so that seems OK. What about a 1 kg ewe? Now we get $-2 + 1.2 \times 1 = -0.8$. Negative lambs! Not a very sensible prediction.

While the professional sheepologists might argue that 1 kg is not a very realistic mass for a sheep, really, we would prefer to have a model that cannot make impossible predictions. This is a job for the link function. When using a GLM, instead of trying to model the predicted values of the response variable directly, *we model a mathematical transformation of the prediction*. The function that does this transformation is called the link function.

Confused? Fair enough, that last paragraph might be a little difficult to understand. Let's work with the Soays again to get a better idea of what the link function does. If we had used a Poisson GLM to model the fitness–mass relationship, then the model for predicted fitness would look like this:

$$\text{Log[Predicted Fitness]} = \text{Intercept} + \text{Slope} \times \text{Body Size}.$$

The important point to note here is that the link function in this case is the *natural log*. The link function in a standard Poisson GLM is always the natural log.

Now, instead of the linear predictor describing fitness directly, it relates the (natural) logarithm of predicted fitness to body size. This has to be

Downloaded from https://academic.oup.com/book/27290/chapter/196938074 by University of Glasgow user on 25 September 2024

positive, but its log-transformed value can take any value we like (7.1, –2, 0, etc.). The log link function means we have to do a tiny bit of algebra to get the equation for the predicted fitness:

$$\text{Predicted Fitness} = e^{\text{Intercept}+\text{Slope}\times\text{Body Size}}.$$

So . . . a Poisson generalized linear model for the Soays actually implies an exponential (i.e. non-linear!) relationship between fitness and body mass. Linear model does not mean linear relationship. Remember that. Incidentally, if you remember the diagnostics for the simple regression model you'll probably realize that this exponential relationship could be a good thing. Just look at Figure 7.1 again too.

In summary, a link function allows us to estimate the parameters of a linear predictor that behaves 'correctly', and it does so by moving us from the response 'scale' to the scale of the linear predictor, in this case the natural log scale, defined by the link function. Phew. Still confused?

We've found that a graphical description of the log link function can help (Figure 7.4). The vertical dashed axis represents the whole real number line, encompassing both positive (top) and negative (bottom). This is where we want to be. However, with count data, we are bounded. . . stuck above 0. This is where the predictions in a Poisson model have to live, i.e. a predicted value has to be greater than zero in order for it to be valid for count data. But to do effective statistics, we need to be on a scale that is unbounded. This is what the link function does. It puts us in a happy

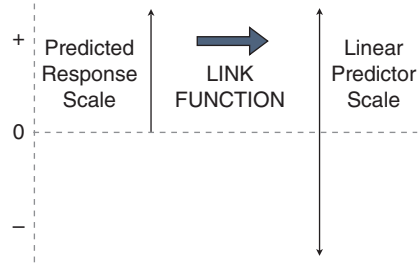

Figure 7.4 Graphical description of the log link function in a GLM.

Downloaded from https://academic.oup.com/book/27290/chapter/196938074 by University of Glasgow user on 25 September 2024

place, where the linear predictor can live and the response can take any value it likes. The logarithm is the 'right' link function to use here, because it moves us from the positive numbers (predicted average counts) to the whole real number line (the linear predictor).

There are various other terms and ideas we could spend time explaining that are a central part of fitting generalized linear models—likelihoods, deviances, likelihood ratio tests, dispersion, etc.—but we've skimmed through enough already to start working with GLMs. This is supposed to be a practical introduction to GLMs in R, so we'll just briefly explain additional aspects as they crop up.

7.4.2 DOING IT RIGHT—ACTUALLY FITTING THE MODEL

Let's see if we can improve the Soay fitness–mass model by fitting a Poisson GLM. The good news is that you already have most of the skills needed to build this model in R and understand it. It is very easy to do this because you've already mastered **lm**(), **autoplot**(), **anova**(), and **summary**(). We are going to use the same *Plot -> Model -> Check Assumptions-> Interpret -> Plot Again* workflow. Since we've already done the *Plot* part, it's time to get on with the rest.

We don't use **lm**() to build a GLM. Instead we use (drum roll...) a function called **glm**(). This works in exactly the same way as **lm**(), though, with the addition that we also have to tell it which family to use:

```
soay.glm <- glm(fitness ~ body.size, data = soay,
                family = poisson)
```

This really is the same as the code for a linear model. We give **glm**() a formula to define the model, some data to work with, and the family to use.

What happened to the link function!? R is very sensible, so if we choose not to specify the link function it will pick a sensible default—the 'canonical link function'. This sounds fancy, but 'canonical' is just another word for 'default'. Remember, this is the log link function for Poisson models. If you want to be explicit about the link function, or change the default, this

Downloaded from https://academic.oup.com/book/27290/chapter/196938074 by University of Glasgow user on 25 September 2024

is easy to do (we'll keep using log). By the way, you can see the canonical links by looking at ?family:

```
soay.glm <- glm(fitness ~ body.size, data = soay,
                family = poisson(link = log))
```

7.4.3 DOING IT RIGHT—THE DIAGNOSTICS

Fitting the GLM was easy. Now, let's look at the diagnostic plots next. These are produced by plotting the model we made, and **autoplot**() can handle a **glm**() just fine. We'll discuss why we use the 'same' diagnostics for the GLM below.

These plots definitely look better (Figure 7.5):

- The plot of residuals vs fitted values (upper left panel) suggests that the systematic part of the model is now pretty good. There is no clear pattern in the relationship, apart from the very slight upward trend at the end. This is nowhere close to being large enough to worry about—it's driven entirely by only two points on the very right.
- The normal Q–Q plot (upper right panel) is also much better. It isn't perfect, as there is some departure from the dashed line, but we shouldn't expect a perfect plot. Life is never perfect. It is a lot better than the corresponding the plot from the **lm**() model, so it looks like our distributional assumptions are OK.
- The scale–location plot (bottom left panel) seems to show a slight positive relationship between the size of the residuals and the fitted values. If you just focus on the points, there isn't much going on.
- The residuals–leverage plot (bottom right panel) is also fine. There is no evidence that we have to worry about outliers or points having too much of an effect on the model.

Is anything bothering you at this point? Perhaps that 'normal Q–Q plot' seems odd. We fitted a Poisson model, so what's all this talk of normal distributions? Perhaps the scale–location plot seems wrong. Surely if the Poisson model was used we should be hoping for a positive relationship, because the variance should increase with the mean?

Downloaded from https://academic.oup.com/book/27290/chapter/196938074 by University of Glasgow user on 25 September 2024

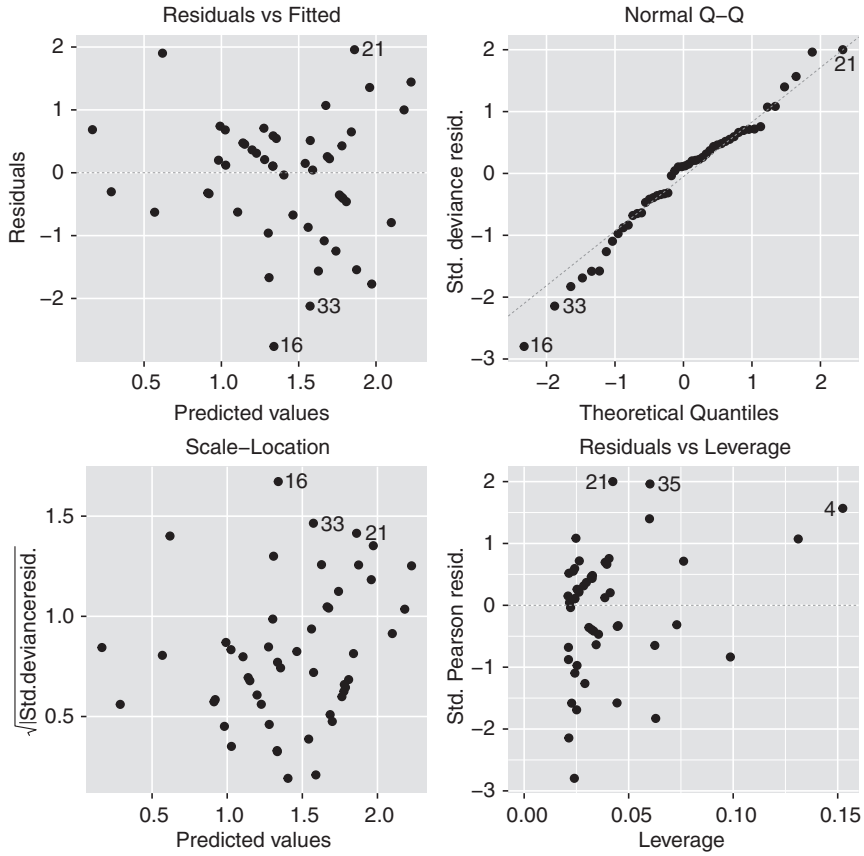

Figure 7.5 Diagnostic graphs for general linear model of fitness–body mass relationship with Poisson error structure and log link function.

Here's the answer(s). When R builds diagnostics for a GLM, it uses something called the standardized deviance residuals. Again, these sound a lot more fancy than they really are. They are a specially transformed version of the raw residuals that make the transformed residuals normally distributed, if (and only if) the GLM family we're using is appropriate.

What this means is, *if the chosen family is a good choice for our data*, then our diagnostics should behave like those from a model with normally distributed errors. You don't need to learn any new skills to evaluate your model with R's GLM diagnostics! The 'normal Q–Q plot' for the transformed residuals *is* checking whether the Poisson distribution is

Downloaded from https://academic.oup.com/book/27290/chapter/196938074 by University of Glasgow user on 25 September 2024

appropriate for the distribution of the residuals, and the scale–location plot *is* checking whether the mean–variance relationship is OK (it will only be patternless if the Poisson distribution is the right model).

7.4.4 DOING IT RIGHT—anova() AND summary()

So far, so good. We have a decent model for the data, so we can finally get back to the original question. Is fitness positively related to body mass? It certainly looks that way, but we still need a *p*-value to convince people. On to step 3 then: testing whether the body mass term is significant.

Once more, you already know how to do this. We use **anova**() (remember, it does not perform an ANOVA). For something produced by **lm**(), **anova**() produces an ANOVA table. Something very similar happens when we use it with a **glm**() model:

```
anova(soay.glm)

## Analysis of Deviance Table
##
## Model: poisson, link: log
##
## Response: fitness
##
## Terms added sequentially (first to last)
##
##
##           Df Deviance Resid. Df Resid. Dev
## NULL                         49     85.081
## body.size  1   37.041        48     48.040
```

The first thing to notice is that it no longer prints an ANOVA table—it shows us an 'Analysis of *Deviance*' table. Don't panic! A lot of this is familiar stuff. The preamble at the top tells you what kind of table it is and reminds you what kind of GLM you used (Poisson, with a log link). It also informs you that this is a sequential table. You know about these. The bit you really care about is that table at the end. We will talk about that in a moment, but first, what is this deviance stuff?

That word 'deviance' is a lot less interesting than it sounds. It is closely related to something called the likelihood, a very general tool for doing statistics. This really isn't the place to explain likelihood, but we'll give you

Downloaded from https://academic.oup.com/book/27290/chapter/196938074 by University of Glasgow user on 25 September 2024

a super-short explanation. Ready? The likelihood of a statistical model, and some data, provides us with a measure of how probable the data would be if they really had been produced by that model. If you know what you're doing, you can use this to find a set of best-fitting model coefficients by picking values that maximize this likelihood thing. Sums of squares, and mean squares allow you to compare different models assuming normality. The likelihood (and deviance) do the same thing for GLMs (and many other kinds of models). OK, that might be the world's shortest introduction to likelihood, but it's good enough for us.

The second thing to notice about the table is that instead of listing sums of squares, mean squares, degrees of freedom, *F*-values, and *p*-values, we only see degrees of freedom, deviances, and residual deviances. Arggghhhh, no *p*-values! The table does tell us that the total deviance in the data (fitness) is 85.081 units and the deviance explained by body size is 37.041 units. That means that nearly half the deviance is accounted for by body size, but what does it mean? Where. Is. Our. *p*-value!?

There's no *p*-value here because R wants you to specify what test to use to calculate it. With a GLM, you can choose different types of tests. Here's what you need to do. *p*-values in the typical GLM involve the χ^2 distribution rather than the *F*-distribution. Note this does not mean we are doing a χ^2 test:

```
anova(soay.glm, test = "Chisq")

## Analysis of Deviance Table
##
## Model: poisson, link: log
##
## Response: fitness
##
## Terms added sequentially (first to last)
##
##
##           Df Deviance Resid. Df Resid. Dev  Pr(>Chi)
## NULL                         49     85.081
## body.size  1   37.041        48     48.040 1.157e-09 ***
## ---
## Signif. codes:
## 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Downloaded from https://academic.oup.com/book/27290/chapter/196938074 by University of Glasgow user on 25 September 2024

With this extra detail in place, we can see that that the test statistic is a χ^2 value of 37.041, with one degree of freedom. The p-value is very small. This is all rather unsurprising given the strong relationship we can see in the data (Figure 7.1). It is good to be sure, though. Where did that p-value come from? It's all about that likelihood again. So, congratulations, you just carried out a likelihood ratio test. That's what **anova**(..., test ="Chisq") is telling R to do. This is highly significant because the deviance associated with the body mass term is so big (trust us on this one). In fact, while you do have to trust us, you can also see that the deviance has dropped from 85 without the body mass explanatory variable to 48 . . . lots of variation has been 'explained'. Does fitness vary positively with ewe body size? Yep ($\chi^2 = 37.04$, $df = 1$, $p < 0.001$)! That's what you might report!

We could actually stop the analysis there. We have analysed our data to test a hypothesis about selection on body size. It looks like there is positive selection operating, so maybe one day Soay sheep will be the size of elephants (or maybe not).

However, we may want to understand what our model is telling us beyond 'there is a significant effect of body mass on fitness'. Let's not forget about the summary table of coefficients. You'll be pleased to find out that we can use **summary**() to print this:

```
summary(soay.glm)

##
## Call:
## glm(formula = fitness ~ body.size, family = poisson(link = log),
##     data = soay)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.7634  -0.6275   0.1142   0.5370   1.9578
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept) -2.42203    0.69432  -3.488 0.000486 ***
## body.size    0.54087    0.09316   5.806 6.41e-09 ***
## ---
## Signif. codes:
## 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Downloaded from https://academic.oup.com/book/27290/chapter/196938074 by University of Glasgow user on 25 September 2024

```
##
## (Dispersion parameter for poisson family taken to be 1)
##
##     Null deviance: 85.081  on 49  degrees of freedom
## Residual deviance: 48.040  on 48  degrees of freedom
## AIC: 210.85
##
## Number of Fisher Scoring iterations: 4
```

This looks familiar . . .

- The first chunk of information at the top is a reminder of the model we are looking at, including information about the family.
- We then see a fairly useless summary of the residuals. Remember, these are those special scaled residuals (the deviance residuals), so a very large number indicates that we have an outlier problem.
- Next come the coefficients. Our model is as simple as they get (it is a line), which means there are only two coefficients: an intercept and a slope. Each estimate has a standard error to tell us how precise it is, a z-value to help us see if the estimate is significantly different from 0, and the associated p-value.
- We are then told about something called the dispersion parameter. This is potentially very important, but now is not the time to explain it. More about this later in this chapter.
- After the dispersion parameter we see summaries of the null deviance, the residual deviance, and their degrees of freedom. The null deviance is a bit like a measure of all the 'variation' in the data. The residual deviance is a measure of what is left over after fitting the model. Big difference => more variation explained.
- Towards the bottom we see the AIC (Akaike information criterion) for the model. We don't use the AIC in this book, but if you are an AIC fan, there's your AIC. Go ahead and use it, for good or for evil.
- You don't need to worry about the 'number of Fisher Scoring iterations'. This is basically a measure of how challenging it was for R to find the best-fit model using all the fancy likelihood stuff.

Downloaded from https://academic.oup.com/book/27290/chapter/196938074 by University of Glasgow user on 25 September 2024

What do those coefficients tell us? Take a moment to digest what you are looking at; look at the coefficients and look at Figure 7.1. When we teach this stuff, we wait for the inevitable penny to drop . . . the intercept . . . it's negative. But a quick look at Figure 7.1 suggests this isn't what we were expecting . . . . So let's ask again . . . what do these coefficients tell us? Well, they *do not* tell us that a 5 kg female will give birth to an average 0.28 lambs over her lifetime ($-2.422 + 0.541 \times 5$). That should be obvious from the figure we made.

What's going on, we ask again? Let's not even once forget about the link function—the GLM is predicting the natural logarithm of lifetime reproductive fitness, *not* the actual fitness. If we want to know how many lambs a 5 kg ewe is predicted to produce, we must account for what the link function did: lifetime fitness = $e^{(-2.422+0.541\times5)} = 1.33$ lambs. Phew. Double phew.

There is one more thing we really should check in the summary output. It has to do with that something we called overdispersion. However, we've got a good rhythm going now, so let's come back this later on. The next part is the fun part—making the publication-ready figure that will make us famous for showing Soay sheep will one day be as large as elephants.

7.4.5 MAKING A BEAUTIFUL GRAPH

The method we'll use to overlay the fitted line on the raw data relies substantially on the generic approach to estimating the line that we outlined in the previous chapter, in the ANCOVA example. You just need to learn two new tricks to make it work for a GLM. As before, we use **expand.grid**() to generate a set of 'new *x*' values, remembering to label the single column name as in the original dataset, i.e. body.size (this is not optional—always do this). Rather than trying to guess where to start and end the 'new *x*' values we'll let the min and max functions do the hard work for us, using the **seq**() function to get 1000 numbers in between the minimum and maximum body mass:

Downloaded from https://academic.oup.com/book/27290/chapter/196938074 by University of Glasgow user on 25 September 2024

```
# note our use of the $ to get the body.size column
min.size <- min(soay$body.size)
max.size <- max(soay$body.size)

# make the new.x values; we use the 'body.size' variable
# name to name the column
# just as it is in the original data.
new.x <- expand.grid(body.size =
                       seq(min.size, max.size, length=1000))
```

Just as with a general linear model, we can now use these 'new *x*'s' with the **predict**() function. Actually, we use **predict**() with three arguments: the GLM model, the value for newdata, and a request for *standard errors*. It is a bit annoying, but **predict**() for **glm**() doesn't have an interval = confidence argument and thus does not return a data frame. You may recall that we asked for interval = confidence in Chapter 6 (in relation to ANCOVA), and that the confidence interval is $\bar{x} \pm 1.96 \times \text{SE}$. We can't get that information automatically. That's why we formally request the standard errors and why we also have to `convert` the output into a data frame using **data.frame**():

```
# generate fits and standard errors at new.x values.
new.y = predict(soay.glm, newdata = new.x, se.fit = TRUE)
new.y = data.frame(new.y)
# check it!
head(new.y)

##         fit    se.fit residual.scale
## 1 0.1661991 0.2541777              1
## 2 0.1682619 0.2538348              1
## 3 0.1703247 0.2534919              1
## 4 0.1723874 0.2531491              1
## 5 0.1744502 0.2528063              1
## 6 0.1765130 0.2524635              1
```

The next step is 'housekeeping'. We need to combine the new *x*'s with the new *y*'s so that everything is in one place. We do that using the **data.frame**() function again. We call the data frame to `addThese` because—just like before—we are going to add these to the plot. And, finally, we mustn't forget one extra *very important change*: we must rename the 'fit' produced by 'predict' to 'fitness' to match the original data. The **rename**() function to the rescue again:

Downloaded from https://academic.oup.com/book/27290/chapter/196938074 by University of Glasgow user on 25 September 2024

```
# housekeeping to bring new.x and new.y together
addThese <- data.frame(new.x, new.y)
addThese <- rename(addThese, fitness = fit)

# check it!
head(addThese)

##   body.size   fitness    se.fit residual.scale
## 1  4.785300 0.1661991 0.2541777              1
## 2  4.789114 0.1682619 0.2538348              1
## 3  4.792928 0.1703247 0.2534919              1
## 4  4.796741 0.1723874 0.2531491              1
## 5  4.800555 0.1744502 0.2528063              1
## 6  4.804369 0.1765130 0.2524635              1
```

Beauty! Now we have a new data frame that contains the grid of body sizes, as well as the predicted fitness values and standard errors at each of these predictions. We are *nearly* ready to plot everything. But how are we going to add those confidence intervals? One more bit of housekeeping will give us what we need. We need to use the **mutate**() function to simultaneously calculate the CIs and include them in `addThese`. This last step effectively generates the data frame that interval = confidence did when we used **predict**() with **lm**():

```
addThese <- mutate(addThese,
                   lwr = fitness - 1.96 * se.fit,
                   upr = fitness + 1.96 * se.fit)
```

OK, now we seem to have everything we need to make our final figure. Let's use the same code format as we used in Chapter 6, and see how it works. But don't study the R code too carefully yet (hint: we might have one more little problem to deal with):

```
ggplot(soay, aes(x = body.size, y = fitness)) +
  # first show the raw data
  geom_point(size = 3, alpha = 0.5) +
  # now add the fits and CIs -- we don't need to specify body.size
  # and fitness as they are inherited from above
  geom_smooth(data = addThese,
              aes(ymin = lwr, ymax = upr), stat = `identity') +
  # theme it
  theme_bw()
```

Downloaded from https://academic.oup.com/book/27290/chapter/196938074 by University of Glasgow user on 25 September 2024

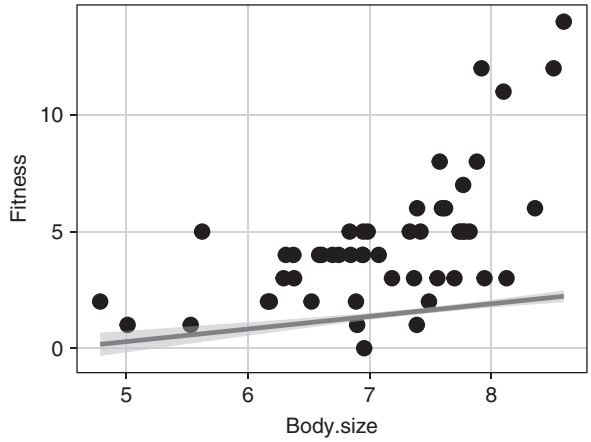

Figure 7.6 Not quite what we wanted. . . the line is clearly not in the right place.

Uh-oh, that did not work as expected (Fig. 7.6). Something is not right—we forgot about the pesky but vital link function again. The default is for **predict**() to produce predictions on 'the scale of the *link function*' (i.e. on the same scale as the linear predictor). Because we have a log link, this means the predictions are the log of the expected fitness. We wanted predictions of the actual fitness. We've shown you how to get it wrong because you will do this at some point. It helps to diagnose the problem if you've seen it before. It helps us daily.

This mistake is easy to fix. All we have to do is apply the inverse of the logarithm to any *y*-axis variables (i.e. 'unlog' them) in addThese. This is the exponential function. We could do this on the fly inside ggplot, but it is less error-prone to modify everything during the process of making addThese. Make sure you understand what we are doing here. This is what the script/code should look like . . .

```
# range of body sizes
min.size <- min(soay$body.size)
max.size <- max(soay$body.size)

# make the new.x values;
# we use the 'body.size' name to name the column
# just as it is in the original data.
```

Downloaded from https://academic.oup.com/book/27290/chapter/196938074 by University of Glasgow user on 25 September 2024

```
new.x <- expand.grid(body.size =
                       seq(min.size, max.size, length=1000))

# generate fits and standard errors at new.x values.
new.y = predict(soay.glm, newdata=new.x, se.fit=TRUE)
new.y = data.frame(new.y)

# check it!
head(new.y)

##         fit    se.fit residual.scale
## 1 0.1661991 0.2541777              1
## 2 0.1682619 0.2538348              1
## 3 0.1703247 0.2534919              1
## 4 0.1723874 0.2531491              1
## 5 0.1744502 0.2528063              1
## 6 0.1765130 0.2524635              1

# housekeeping to bring new.x and new.y together
addThese <- data.frame(new.x, new.y)

# check it!
head(addThese)

##   body.size       fit    se.fit residual.scale
## 1  4.785300 0.1661991 0.2541777              1
## 2  4.789114 0.1682619 0.2538348              1
## 3  4.792928 0.1703247 0.2534919              1
## 4  4.796741 0.1723874 0.2531491              1
## 5  4.800555 0.1744502 0.2528063              1
## 6  4.804369 0.1765130 0.2524635              1

# exponentiate the fitness and CI's to get back the
# 'response' scale
# note we don't need rename() because mutate() works with
# the fit values each time, and we 'rename' inside mutate()
addThese <- mutate(addThese,
                   fitness = exp(fit),
                   lwr = exp(fit - 1.96 * se.fit),
                   upr = exp(fit + 1.96 * se.fit))
 # check it!
head(addThese)

##   body.size       fit    se.fit residual.scale  fitness
## 1  4.785300 0.1661991 0.2541777              1 1.180808
## 2  4.789114 0.1682619 0.2538348              1 1.183246
## 3  4.792928 0.1703247 0.2534919              1 1.185690
## 4  4.796741 0.1723874 0.2531491              1 1.188138
## 5  4.800555 0.1744502 0.2528063              1 1.190591
## 6  4.804369 0.1765130 0.2524635              1 1.193050
##         lwr      upr
```

Downloaded from https://academic.oup.com/book/27290/chapter/196938074 by University of Glasgow user on 25 September 2024

```
## 1 0.7174951 1.943300
## 2 0.7194600 1.946004
## 3 0.7214303 1.948712
## 4 0.7234059 1.951425
## 5 0.7253869 1.954141
## 6 0.7273732 1.956861

# now the plot on the correct scale
ggplot(soay, aes(x = body.size, y = fitness)) +
  # first show the raw data
  geom_point(size = 3, alpha = 0.5) +
  # now add the fits and CIs -- we don't need to specify
  # body.size and fitness as they are inherited from above
  geom_smooth(data = addThese,
              aes(ymin = lwr, ymax = upr), stat = 'identity') +
  # theme it
  theme_bw()
```

This time our ggplot() efforts (Figure 7.7) work as hoped, because everything is on the right scale.

There seems to be a lot going on in that code, but you've seen it all before. The usual raw data-plotting syntax is in there (geom_point()), along with geom_smooth() to add the fits and CIs. Just as with the ANCOVA example, we have to provide geom_smooth() with some new aesthetics for the CIs, but it inherits the 'x =' and 'y =' aesthetics.

Good work. It really isn't all that much effort to produce a sophisticated summary figure from GLM-worthy data in R. It is mostly like plotting

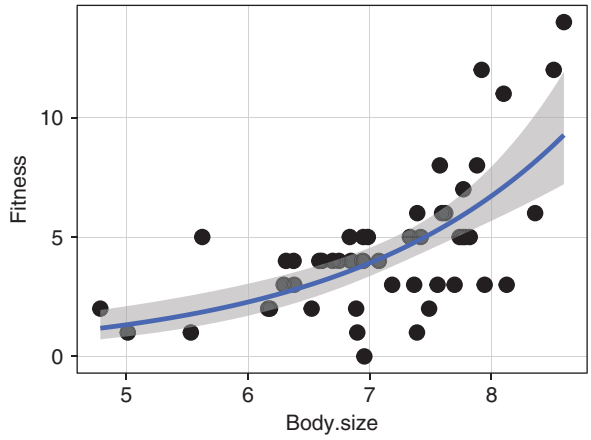

Figure 7.7 The line is now in the correct place. Great!

Downloaded from https://academic.oup.com/book/27290/chapter/196938074 by University of Glasgow user on 25 September 2024

the results of a linear regression or ANCOVA—just be careful with your housekeeping—as long as you remember the added twist introduced by the link function and the need to 'back transform' from the scale of the linear predictor.

7.5 When a Poisson GLM isn't good for counts

You've now seen all the pieces of the GLM puzzle. We've talked about the basic theory, how to build a GLM in R, interpreting diagnostic plots, making sense of the output from **anova**() and **summary**(), and, finally, how to construct the ever-important figure for publication. We made it look pretty easy too. We did all of this using the simplest possible Poisson regression, and we used data that were very well behaved. It was almost like they were too good to be real data. . . What happens when things are not too good to be true . . .

7.5.1 OVERDISPERSION

It would be very irresponsible to teach you about GLMs in R without discussing the dreaded 'overdispersion'. Overdispersion is just a fancy stats word for 'extra variation'. We're going to explain what this means in the real world, describe some common causes of overdispersion, and then tell you why these matter. We'll finish up by showing you how to detect overdispersion and suggest a couple of possible quick fixes to help you deal with it.

Some GLMs make very strong assumptions about the nature or variability in your data. The Poisson GLM is one such example, as is the binomial GLM. Earlier we noticed that the variance of the Poisson distribution increases with the mean. Actually, the variance *is exactly equal to* the mean for a Poisson distribution. This assumption is only likely be true of real Poisson count data if we are able to include every single source of variation in our analysis.

This is a big assumption, particularly in biology. When you go outside and measure the various features of individuals, populations, or whatever

Downloaded from https://academic.oup.com/book/27290/chapter/196938074 by University of Glasgow user on 25 September 2024

else it is you're studying, no matter how hard you work, there will always be important aspects of your 'things' you can't measure. When these influence your count variable, they create variation you can't account for with a standard Poisson GLM. Even under carefully controlled lab conditions there will be sources of variation you cannot control. Maybe you put all your insects into a controlled environment, but did you ensure they all had the same body size and experienced exactly the same rearing environment? (If you did, well done; that was a lot of work.)

'Ignored stuff' is the root source of the overdispersion problem, but does it arise only from excluded variables? Sadly not. It arises when there is *non-independence* in the data. What does this mean? It means some of your 'things' in your data—individuals, populations, etc.—are more similar to one another than they are to other 'things'. For example, in that insect experiment, individuals from the same clutch/brood/family will be more similar to one another than to a randomly chosen insect, because they share genes and a common rearing environment. We could go on and on about non-independence, but you get the idea.

You might be thinking, 'So what? All models are wrong. . . ' The problem with overdispersion is that if it's there, and you ignore it, it messes up the precious *p*-values. And it does it the worst possible way—they will be anti-conservative. Arghhhhhh! Wait, what does that mean? This is a very serious problem. It means you end up with more false positives than you should when the stats are 'working' properly, and *that* means you'll think you've found evidence for things happening that are not in fact happening.

OK, overdispersion is probably common and it's very naughty to ignore it because it breaks *p*-values. What can you do about it? Your first job is to detect it. The good news is that this is reasonably easy to do. We just need to know about a new diagnostic trick.

Look at the summary of the Soay sheep Poisson regression again:

```
summary(soay.glm)

##
## Call:
```

Downloaded from https://academic.oup.com/book/27290/chapter/196938074 by University of Glasgow user on 25 September 2024

```
## glm(formula = fitness ~ body.size, family = poisson(link = log),
##     data = soay)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.7634  -0.6275   0.1142   0.5370   1.9578
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept) -2.42203    0.69432  -3.488 0.000486 ***
## body.size    0.54087    0.09316   5.806 6.41e-09 ***
## ---
## Signif. codes:
## 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for poisson family taken to be 1)
##
##     Null deviance: 85.081  on 49  degrees of freedom
## Residual deviance: 48.040  on 48  degrees of freedom
## AIC: 210.85
##
## Number of Fisher Scoring iterations: 4
```

The bit we care about here is near the bottom: the 'Residual deviance' (48.040) and its degrees of freedom (48). If a GLM is working perfectly and there is no overdispersion, these two numbers should be equal. The GLM output even declares this constraint, stating 'Dispersion parameter for poisson family taken to be 1'. We can calculate a kind of 'dispersion index' by dividing the residual deviance by the residual degrees of freedom (don't mix up the order of division). This should be about 1—if it is much bigger, the data are overdispersed, and if it is much less, they are underdispersed (which is rare). For our Soay model, there is nothing to worry about as the residual deviance and its degrees of freedom are almost identical.

What if the dispersion index had been 1.2, 1.5, 2.0, or even 10? When should you start to worry about overdispersion? That is a tricky question to answer. One common rule of thumb is that when the index is greater than 2, it is time to start worrying. Like all rules of thumb, this is only meant to be used as a rough guide. In reality, the worry threshold depends on things like sample size and the nature of the overdispersion. If in doubt,

Downloaded from https://academic.oup.com/book/27290/chapter/196938074 by University of Glasgow user on 25 September 2024

ask someone who knows about statistics to help you decide whether or not to worry. Alternatively, you could try out a different kind of model.

What are your options?

- One simple way to fix a model indicating overdispersion is by changing the family in **glm**() to a 'quasi' version of your distribution (e.g. `family = poisson` becomes `family = quasipoisson`). A 'quasi' model works exactly like the equivalent standard GLM model, but it goes one step further by estimating that dispersion index we mentioned above—it does it in a more clever way than we did. Once this number is known, R can adjust the *p*-values to account for it.
- Another simple way to fix an overdispersed Poisson model is to switch to the 'negative binomial' family. A negative binomial distribution can be thought of as a more flexible version of the Poisson distribution. The variance increases with the mean for a negative binomial distribution, but it does so in a less constrained way, i.e. the variance does not have to equal the mean.

Let's discuss briefly how these two options work in the real world.

A 'quasi-' model works exactly like the equivalent standard GLM model. If you build two versions of the Soay model—one using `family = poisson`, the other using `family = quasipoisson`—and compare the summary tables, you will see that the coefficient estimates are the same in both models (go on, try it). The only thing that should be different is 'the stats'. These are based on a method that accounts for the overdispersion (or underdispersion).

Simple, yes? Well, there is one more thing to be aware of. You should be careful when you use anova with a 'quasi' model, because you have to explicitly tell R to take account of the estimated dispersion. This is not hard to do. Instead of using **anova**(..., test = "Chisq"), you have to use **anova**(..., test = "F"). This tells R to use an *F*-ratio test instead of a likelihood ratio test, because this allows us to incorporate the dispersion estimate into the test. We won't say more, as there is too much statistical magic here to explain in a simple way.

Downloaded from https://academic.oup.com/book/27290/chapter/196938074 by University of Glasgow user on 25 September 2024

Negative binomial GLMs are easy to work with. However, we *do not* use **glm**() to build these models; use **glm.nb**() from the ***MASS*** package instead. The ***MASS*** package is part of base R, so you won't need to install this; just use ***MASS***. Other than this, there is very little new about using negative binomial GLMs. You don't need to worry about the family, because **glm.nb**() deals only with the negative binomial. There is a link function to worry about—the default is the natural log again, but you have a couple more options (take a look at `?glm.nb`) —and, of course, you have to use R's fantastic formula to define the model.

That's it for 'quasi' and negative binomial models. We could walk you through how to use them, but, really, you already have all the skills you need to do this on your own.

We'll finish off with a little warning, though. The 'quasi' and negative binomial tricks often work well when overdispersion is produced by missing variables. Sadly, they don't really fix overdispersion that is generated by non-independence. That kind of problem is better dealt with using more sophisticated models. This isn't the place to discuss them, but just in case you're the kind of person that *needs* to know, we'll tell you what the most common solution is called: a mixed model.

7.5.2 ZERO INFLATION

There's one very specific source of overdispersion in Poisson-like count data that's worth knowing about if you're a biologist: zero inflation. This happens when there are too many zeros relative to the number we expect for whatever distribution we're using. If your count is zero-inflated, you can often spot it with a bar chart of raw counts. If you see a spike at 0, that's probably (not always!) caused by zero inflation.

Biological counts are often zero-inflated. This often results from a binary phenomenon acting in combination with a Poisson process. For example, the number of fruits produced by an outcrossing plant depends first on whether it is ever visited by a pollinator, and then on the number of flowers visited. The number of zeros could be very high if there are few pollinators around because many plants are never visited, but for

Downloaded from https://academic.oup.com/book/27290/chapter/196938074 by University of Glasgow user on 25 September 2024

those plants that are visited, the number of seeds might be nicely Poisson distributed.

Zero inflation is best dealt with by using a new model. There are (at least) a couple of options available here:

- Option 1 is to use a *mixture model*. These work by modelling the data as a mixture of two distributions (sometimes statisticians do give things sensible names). They assume each observation in the data comes from one of these two distributions. We don't know which, but, fortunately, some clever statistical machinery can deal with this missing information.
- Option 2 is to use a *hurdle model*. A hurdle model has two parts. It has a binary part for the zeros (it asks whether the runner makes it over the hurdle—zero or not), and a Poisson part for the non-zero values (it asks how many steps they then take—a positive integer). The clever part is that the Poisson part is modified so that it only allows for positive values.

If you do run into a zero-inflation situation, R has you covered (of course). There are quite a few options available, but the most accessible is probably the ***pcsl*** package (available on CRAN). This provides a couple of functions—**zeroinfl**() and **hurdle**()—to model zero-inflated data using a mixture or hurdle model.

7.5.3 TRANSFORMATIONS AIN'T ALL BAD

You probably learned in your first stats course that you should always transform response data that are counts, and then use a general linear model. The log and square root transformations are the usual options for Poisson-like count data. Later on, if you did a more advanced course that included GLMs, you may have been told never to transform your count data. Instead, use a GLM, because. . . they're just better.

Who is right? We think they are both wrong. Sometimes transformations work, sometimes they don't. Sorry, we're pragmatists, which means

Downloaded from https://academic.oup.com/book/27290/chapter/196938074 by University of Glasgow user on 25 September 2024

we are OK with shades of grey. If you've designed a good experiment, you probably just want to know whether a treatment had an effect or not (*p*-values), and whether that effect is big or not (coefficients). If in doubt, try out a transformation, build the model, and check the diagnostics. If these look fine, it's probably OK to use the model. There *are* some advantages of using transformations:

- Transformations often work just fine when the data are 'far away from zero' (so there are no zeros) but don't span many orders of magnitude.
- Using a transformation is simple because there are no nasty link functions to worry about. But. . . you can also analyse fancy experiments like 'split-plot' designs fairly easily.
- You don't have to worry about overdispersion. The residual error term takes care of the overdispersion. This can be a big advantage.

So why don't we always use transformations? Sometimes you just can't find a good one:

- Transformations change two things at the same time. They alter the mean–variance relationship, and they change the 'shape' of the relationship between predictor variables and the response. A transformation might fix one, but break the other.
- Those zeros again. Transformations often fail when your count data contain zeros. You can't take the log of zero, so you have to use $\log(y + 1)$. This almost never produces a model with good diagnostics.
- The model you build may be difficult to interpret and use because it does not make predictions on the scale you originally measured things on.

If you aren't sure what to do, explore some options and follow the approach that we advocate in this book: *Plot -> Model -> Check Assumptions*.

Downloaded from https://academic.oup.com/book/27290/chapter/196938074 by University of Glasgow user on 25 September 2024

7.6 Summary, and beyond simple Poisson regression

Well, that was fun, don't you think? You've just been introduced to the GLM. We've discussed data types that often necessitate (i.e. beg for) a GLM, including count data, binary data. We have deliberately introduced more theory here, as a way to understand general versus generalized linear models, and also to boost your confidence. Statistical modelling is not easy, but you have embarked or are just embarking on a career in research, and understanding deeply how to manage, communicate, and analyse your data is vital. You now know a great deal. Perhaps just enough to be *dangerous*.

7.6.1 THE LINK FUNCTION RULES

We hope, perhaps more than anything, that you now understand that when we specify a family to manage 'non-normal' data, we are making very specific assumptions about the nature of the variability in those data. This is very useful, because knowing these assumptions allows us to use all of the diagnostics and tools we are accustomed to from general linear models, even though things like odd mean–variance relationships are in play. The downside is that sometimes our data won't play ball with these assumptions. Overdispersion is a common problem in these cases.

We also introduce a link function into the mix whenever we build a GLM. The link function manages the fact that predictions for many types of data must be bounded. A model that predicts negative counts is not ideal, for example. We refer you again to Figure 7.4. If you understand that we have to move from the response scale to the scale of the link function to fit the model, and then back again to interpret and plot the results, you have won the game. Now you can read the hard books.

7.6.2 THE WORKFLOW STAYS THE SAME

In Chapters 5 and 6, we introduced a deliberate and recipe-like workflow of *Plot -> Model -> Check Assumptions -> Interpret -> Plot Again*. This

Downloaded from https://academic.oup.com/book/27290/chapter/196938074 by University of Glasgow user on 25 September 2024

workflow does not change with a GLM. Some bits are harder. Making the first and last figures can require more thinking. Luckily, when used appropriately, interpreting the diagnostic plots remains as straightforward as ever. Interpreting the **anova**() and **summary**() tables requires understanding a bit more about likelihood and deviance, but, at the same time, the structure and methods for generating and interpreting these tables remain the same. As the models you fit become more complicated (e.g. interactions between variables), the **anova**() table is still telling you about each part of the model (main effects and interactions), treatment contrasts are still the default in the **summary**() table, and alphabetical listing of categorical-variable levels is still the norm.

7.6.3 BINOMIAL MODELS?

We have not covered binomial models, which, as we note, come in many forms. We refer you now to more advanced books on the GLM, including a bible of the R community, *Modern Applied Statistics with S*. (see Appendix 2). What you will find in your readings, and in your exploration of various web resources, is that the model formula works just as it does in **lm**(), as with our Poisson GLM, but that there are several different ways of coding the response variable—it can be binary, percentages with weights, or a freaky two-column variable of success/failure from trials. Oh, and the default link function is the logit link, with a couple more options if you need them. We have to use a new link function because the goal of a binomial GLM is to model probabilities of events, and these have to lie between 0 and 1. Don't let these things scare you, though. You *are* more than ready now to tackle more advanced readings and ideas.

Downloaded from https://academic.oup.com/book/27290/chapter/196938074 by University of Glasgow user on 25 September 2024