Processing and Analysis of Biological Data

Generalized Linear Models (GLM)

Øystein H. Opedal

14 Nov 2022

Introduction

Generalized linear models (GLMs) add flexibility to the linear model by allowing deviations from the usual assumption of normally distributed residuals. Briefly, a GLM consists of the familiar linear predictor of a linear model (often denoted as η),

$\eta = \beta_0 + \Sigma_j x_{ij}\beta_j + \epsilon_i$

and a link function, g, that places the predictor on a Gaussian (normally-distributed) scale.

$$y = g^{-1}(\eta)$$

Before going into details about GLMs, we need to recall some basics about the most common error distributions used in biological data analyses.

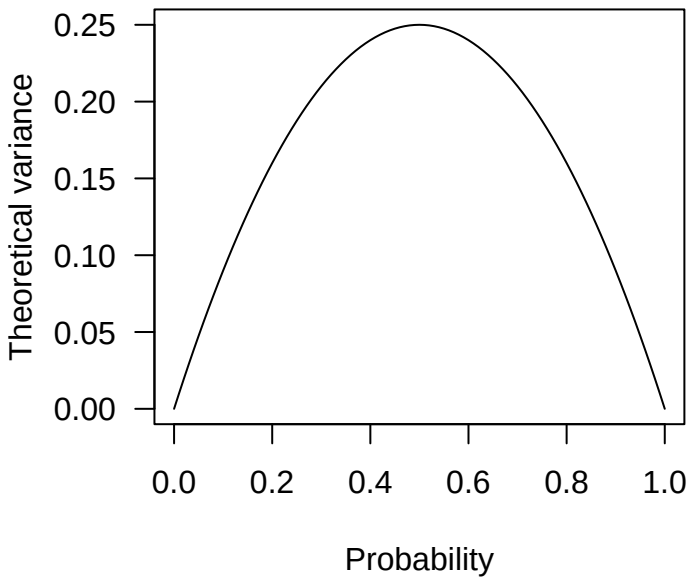
Mean-variance relations for the binomial and poission distributions

The binomial distribution has two parameters, n and p, and summarizes a set of so-called Bernoulli trials with two possible outcomes, yes (1) or no (0). When we have performed more than one trial, we can compute the proportion p of the n trials with a positive outcome.

The theoretical variance of the binomial distribution is given by

$\sigma^2 = np(1-p)$

```
#rbinom(3, 10, c(0.1, 0.5, 0.9))

x = seq(from=0, to=1, by=0.01)
v_b = x*(1-x) #Binomial variance
plot(x, v_b, type="l", xlab="Probability", ylab="Theoretical variance", las=1)
```



This is important to keep in mind, because it affects how we can compare the (proportional) variation of variables measures as proportions. If e.g. one population has a mean of 0.5, if will be expected to be much more variable than a second population with a mean of 0.1, just because there is less opportunity to vary. This is a now well-recognized problem e.g. in demography research, where the interest is often in comparing the extent of variation in life-history traits measured as proportions, such as germination or survival. One proposed solution is to scale the observed CV by the maximum based on the theoretical variance, but a perhaps simpler approach is to transform the proportional data in a way that makes them approach a normal distribution.

One previously popular but now not recommended transformation is the so-called arcsin square-root transformation,

$x' = \arcsin(\sqrt{x})$

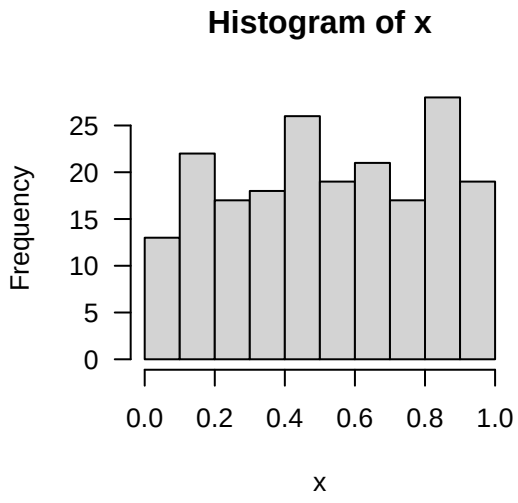
A more meaningful transformation is the logit or log odds transformation

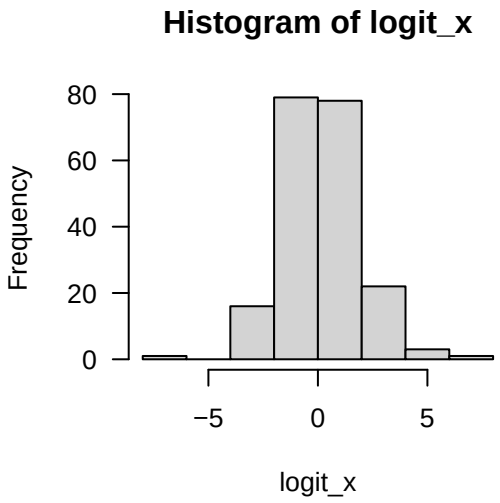
$logit(x) = log(\frac{x}{1-x})$

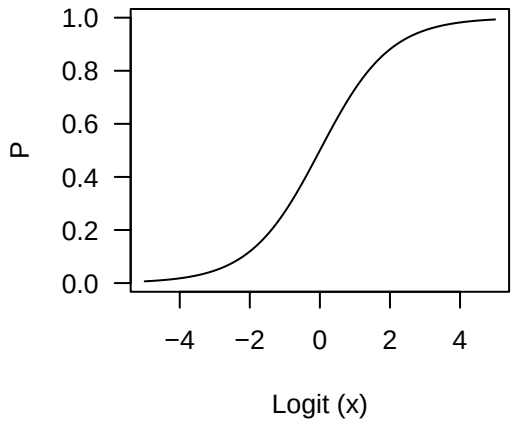
In the following example we are using some functions. This may be a good time to visit the Appendix introducing some basics on writing functions. Note that when functions are given on a single line, we can skip the special curvy brackets (`{`).

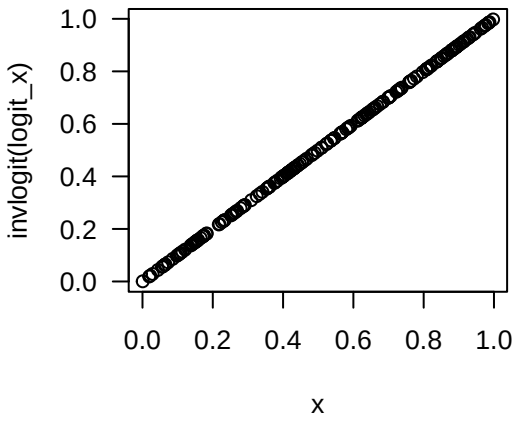
```
logit = function(x) log(x/(1-x))
invlogit = function(x) 1/(1+exp(-x))

x = runif(200)
logit_x = logit(x)

par(mfrow=c(2,2))
hist(x, las=1)
hist(logit_x, las=1)
```

```
xx = seq(-5, 5, 0.01)
plot(xx, invlogit(xx), type="l", las=1,
     xlab="Logit (x)",
     ylab="P")
plot(x, invlogit(logit_x), las=1)
```



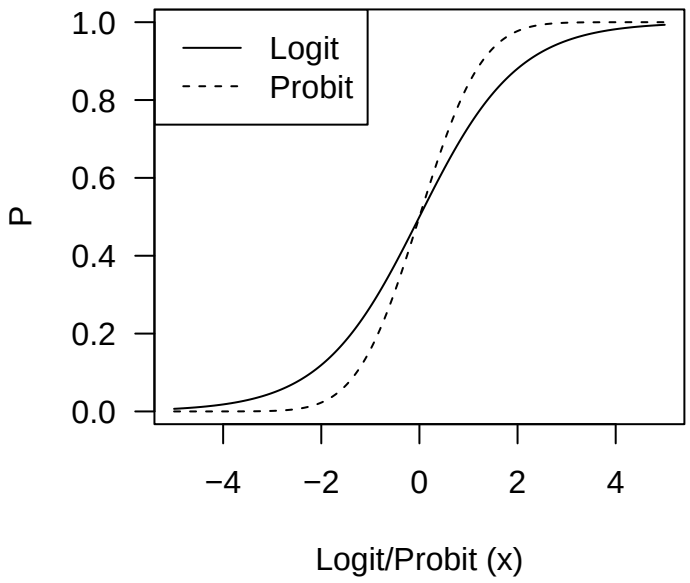






The logit transformation is the most common *link function* in a Generalized Linear Model with binomial errors. A similar alternative is the so-called probit link, which corresponds to the quantile distribution of the standard normal distribution (which is why we can use the `pnorm` function to compute the inverse).

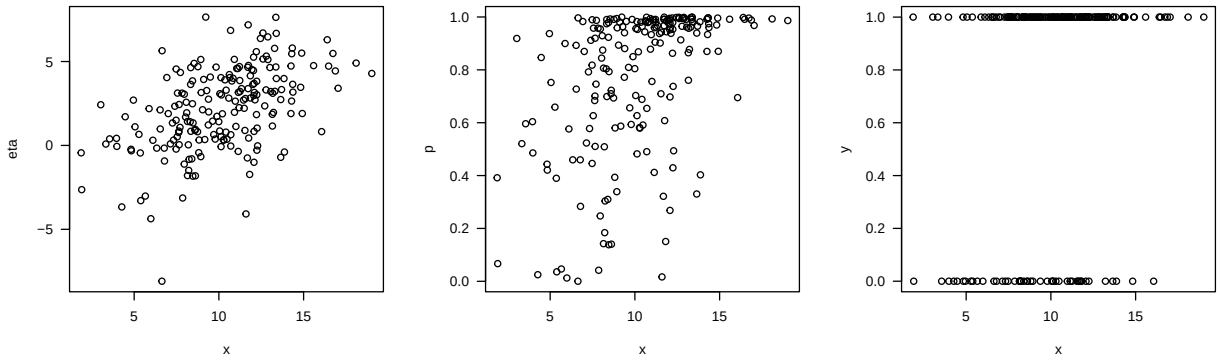
```
plot(xx, invlogit(xx), type="l", las=1,
     xlab="Logit/Probit (x)",
     ylab="P")
lines(xx, pnorm(xx), lty=2)
legend("topleft", legend=c("Logit", "Probit"),
       lty=c(1,2))
```



Logistic regression

As we have already seen, a logistic regression (GLM with binomial errors) is well suited for analysing binary data (or proportions).

```
x = rnorm(200, 10, 3)
eta = -2 + 0.4*x + rnorm(200, 0, 2)
p = invlogit(eta)
y = rbinom(200, 1, p)

par(mfrow=c(1,3))
plot(x, eta, las=1)
plot(x, p, las=1)
plot(x, y, las=1)
```



Above, we simulated data by first formulating a linear predictor η, then transforming the predicted values into probabilities (through the inverse logit transformation), and finally binarizing the data by sampling from the binomial distribution. The last step adds additional uncertainty by accounting for the stochasticity of the observation process.

```
m = glm(y~x, family=binomial(link="logit"))
summary(m)
```

```
##
## Call:
## glm(formula = y ~ x, family = binomial(link = "logit"))
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.1762   0.3644   0.6225   0.7891   1.3062
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept) -0.63658    0.55738  -1.142  0.25341
## x            0.18091    0.05675   3.188  0.00143 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 222.71  on 199  degrees of freedom
## Residual deviance: 211.73  on 198  degrees of freedom
## AIC: 215.73
##
## Number of Fisher Scoring iterations: 4
```

The summary table includes, as always, a lot of information. The first thing to be aware is that when we are fitting a GLM, we obtain the parameter estimates on the link scale (here logit). Note that the parameter estimates are not too far from those we used to define the linear predictor η when we simulated the data. These values are meaningful as such, and if the predictor variable has units of *mm*, the slopes have units of *log odds* mm^{-1}.

To interpret the results biologically and to represent them in graphs, it can be useful to backtransform the predicted values to the probability scale. For example, we can ask how much the probability changes for a

standard deviation increase in the predictor variable (though note that this is no longer a linear transform, so the consequences of increasing and decreasing the predictor by 1 standard deviation may be different).

Recall from the previous section that a log odds of 0 corresponds to a probability of 0.5 ($log(\frac{0.5}{1-0.5}) = log(1) = 0$). If we solve the model equation (the linear predictor) for 0, we can thus obtain the predictor value corresponding to a probability of 0.5, which is often a relevant benchmark.

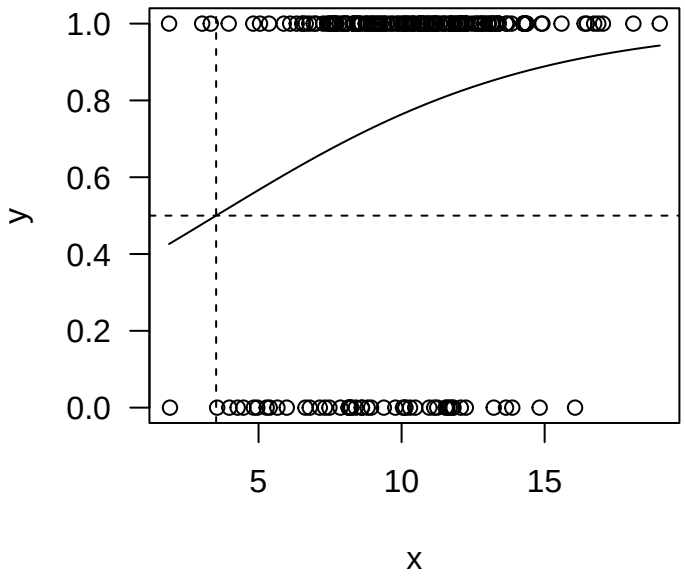
$0 = \beta_0 + \beta_x$

$\frac{-\beta_0}{\beta_x} = x$

EXERCISE: Replicate the plot below. To produce a regression line, we define some new x-values that spans the data range along the y-axis, then obtain predicted values $\hat{y}$ (using the model coefficients), and finally transform these values to the probability scale to obtain the predicted probabilities $\hat{p}$.

```
coefs = summary(m)$coef

x_pred = seq(from=min(x), to=max(x), by=0.01)
y_hat = coefs[1,1] + coefs[2,1]*x_pred
p_hat = invlogit(y_hat)
```

Compute the value of x corresponding to a probability of 0.5, and add lines to the plot to illustrate the results.



The GLM summary table does not provide an r^2 value, because the normal r^2 does not work for logistic regression. There are however several 'Pseudo-r^2' available, typically based on comparing the likelihood of the model to that of a null model (a similar model but with only an intercept). The `MuMIn` package provides one such measure.

```
library(MuMIn)
r.squaredGLMM(m)
```

```
## Warning: 'r.squaredGLMM' now calculates a revised statistic. See the help page.
```

```
## Warning: the null model is correct only if all variables used by the original
## model remain unchanged.
```

```
##                    R2m        R2c
## theoretical 0.08886671 0.08886671
## delta       0.05602833 0.05602833
```

Another possibility for logistic regression is to compute the coefficient of discrimination, or Tjur's D. Indeed, there are several ways to evaluate the performance of a statistical model, and in a logistic regression it makes sense to ask how well the model discriminates between true positives and negatives in the data.

The coefficent of discrimination is defined as $D = \bar{\hat{\pi}_1} - \bar{\hat{\pi}_0}$, and is computed by comparing the predicted probability for those data points that are successes or positives (1s; $\bar{\hat{\pi}_1}$) to the predicted probability for those data points that are failures or negatives (0s; $\bar{\hat{\pi}_1}$).

```
y_hat = coefs[1,1] + coefs[2,1]*x
p_hat = invlogit(y_hat)

mean(p_hat[which(y==1)]) - mean(p_hat[which(y==0)])
```

```
## [1] 0.05628918
```

Some final notes on fitting binomial GLM's. There are three ways to formulate these models in R. In the example above, the data were 0's and 1's, and we could specify the model simply as

```
glm(y ~ x, family=binomial(link="logit"))
```

When each observation is based on more than one trial, we can formulate the model in two ways. The first is

```
glm(y ~ x, family=binomial(link="logit"), weights=n)
```

where `y` is the proportion of successes, and `n` is the number of trials. The second method is to fit a two-column matrix as response variable, where the first colomn is the number of successes, and the second column is the number of failures, i.e. `y = cbind(successes, failures)`. The model formula is then

```
glm(cbind(successes, failures) ~ x, family=binomial(link="logit"))
```

Data exercise: seed germination

The following data are from a study investigating patterns of seed dormancy in a plant. In this plant species, seeds needs to stay in dry conditions for a specific amount of time before they are ready to germinate, a process known as 'after-ripening'. once the seeds are 'ripe' they will normally germinate when exposed to favourable (moist) conditions.

After different durations of after-ripening, the seeds were sown on moist soil and their germination success (proportion of seeds germinated) recorded. The seeds came from four different populations, and were weighed (in mg) prior to sowing.

The variables in the data are as follows:

- `pop` = Population ID
- `mother` = Maternal plant ID

- `crossID` = Unique cross identifier
- `blocktray` = Sowing tray (experimental block)
- `timetosowing` = Time in days from seed dispersal to watering
- `MCseed` = Population-mean-centered seed mass in mg
- `nseed` = Number of seeds sown
- `germ2` = Proportion of seeds germinated

Analyse the data to estimate the pattern of germination success in response to variation in the duration of after-ripening. Are the patterns similar in different populations? Are there other factors affecting germination success? Produce relevant summary statistics, parameter estimates, and graphs.

```
dat = read.csv("datasets/dormancy/dormancy.csv")
names(dat)
```

```
## [1] "pop"          "mother"       "crossID"      "blocktray"    "timetosowing"
## [6] "MCseed"       "nseed"        "germ2"
```

As a suggested start, the following lines fit a simple model to data from one population using two different methods. Note that the model fit is the same (the log Likelihood of the two models is identical).

```
subdat = dat[dat$pop=="CC",]

germ = subdat$germ2 * subdat$nseed #Successes
notgerm = subdat$nseed - germ #Failures

mod1 = glm(cbind(germ, notgerm) ~ timetosowing, "binomial", data=subdat)
mod2 = glm(germ2 ~ timetosowing, "binomial", weights=nseed, data=subdat)
logLik(mod1) == logLik(mod2)
```

```
## [1] TRUE
```

Can you use the fitted models to estimate the duration of after-ripening required for the expected germination rate to be 0.5?

Suggested analysis

We can use the data to ask how the probability of germination depends on the duration of after-ripening, and whether this further depends on seed size. To do so, we fit a model including both after-ripening time and seed size as predictors.

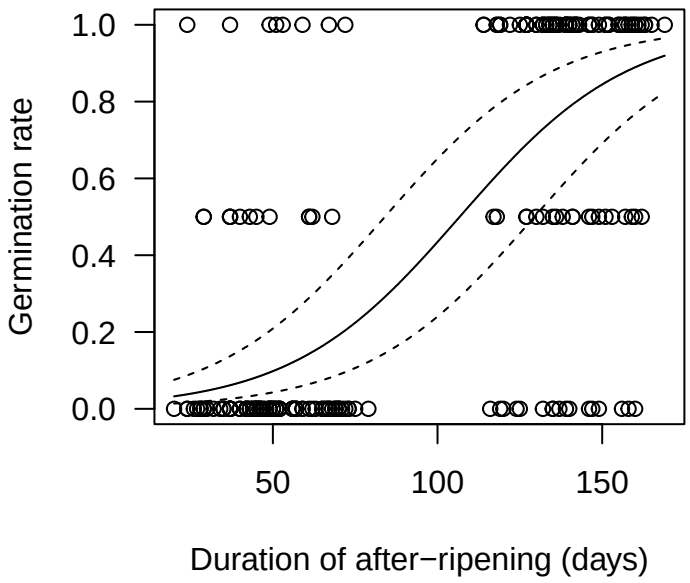
```
mod3 = glm(germ2 ~ timetosowing + MCseed, "binomial", weights=nseed, data=subdat)
summary(mod3)
```

```
##
## Call:
## glm(formula = germ2 ~ timetosowing + MCseed, family = "binomial",
##     data = subdat, weights = nseed)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.7999  -0.7108  -0.4028   0.8715   3.3335
##
```

```
## Coefficients:
##               Estimate Std. Error z value Pr(>|z|)
## (Intercept)  -4.175210   0.369903 -11.287  < 2e-16 ***
## timetosowing  0.039120   0.003308  11.825  < 2e-16 ***
## MCseed       -0.217828   0.035230  -6.183 6.29e-10 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 536.70  on 230  degrees of freedom
## Residual deviance: 293.46  on 228  degrees of freedom
## AIC: 345.2
##
## Number of Fisher Scoring iterations: 5
```

The model suggests detectable effects of both time and seed size. As always with GLMs, it is a little hard to judge the strength of the effects directly from the parameter estimates, because these are on the link scale (here logit). A figure will often help. To illustrate the effect of seed size, we add regression lines corresponding to the mean seed size (solid line), as well as an increase or decrease in seed size by one standard deviation (dashed lines).

```
plot(subdat$timetosowing, subdat$germ2,
     xlab="Duration of after-ripening (days)",
     ylab="Germination rate", las=1)
xvals = seq(min(subdat$timetosowing, na.rm=T),
            max(subdat$timetosowing, na.rm=T), 0.01)

coefs = summary(mod3)$coef
y_hat = coefs[1,1] + coefs[2,1]*xvals
lines(xvals, invlogit(y_hat))

y_hat2 = coefs[1,1] + coefs[2,1]*xvals + coefs[3,1]*sd(subdat$MCseed)
lines(xvals, invlogit(y_hat2), lty=2)

y_hat3 = coefs[1,1] + coefs[2,1]*xvals - coefs[3,1]*sd(subdat$MCseed)
lines(xvals, invlogit(y_hat3), lty=2)
```



To calculate the duration of after-ripening needed for a 50% germination rate, we use the equation above to find that this would be 106.7 days in this population.

```
-coefs[1,1]/coefs[2,1]
```

```
## [1] 106.7274
```

To quantify the seed size effect, we can ask how this changes for a seed that is one standard deviation larger or smaller than the mean.

```
-(coefs[1,1] + coefs[3,1]*sd(subdat$MCseed))/coefs[2,1]
```

```
## [1] 129.424
```

```
-(coefs[1,1] - coefs[3,1]*sd(subdat$MCseed))/coefs[2,1]
```

```
## [1] 84.03079
```

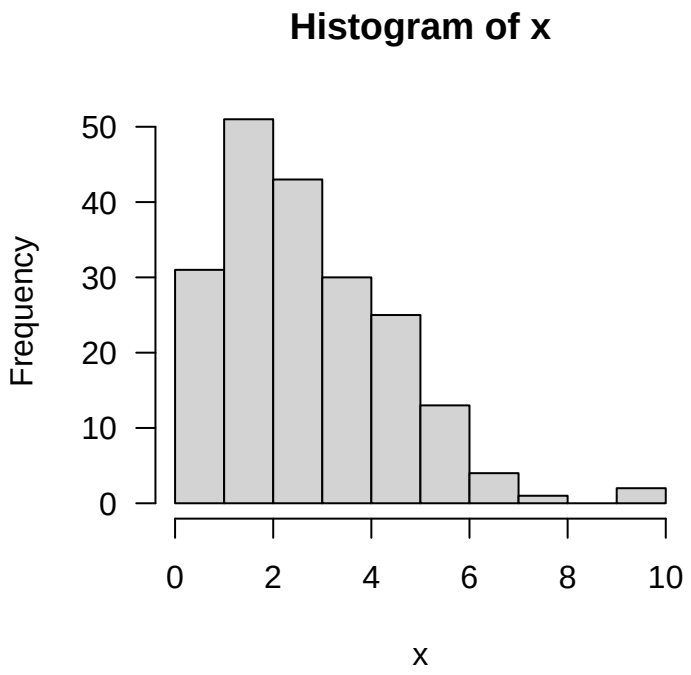
We could write the results like this: The probability of germination increased with longer duration of after-ripening (Fig. 1, Table 1). A seed of average size would have 50% probability of germinating when sown after 106.7 days of after-ripening. For a seed one standard deviation larger or smaller than the mean, this period would change to 129.4 days and 84.0 days, respectively.

Poisson and negative-binomial regression

A second very common data type in biology is count data, which occurs when we have counted something. In ecological studies we often count individuals or species, and in evolutionary biology we often count e.g. the

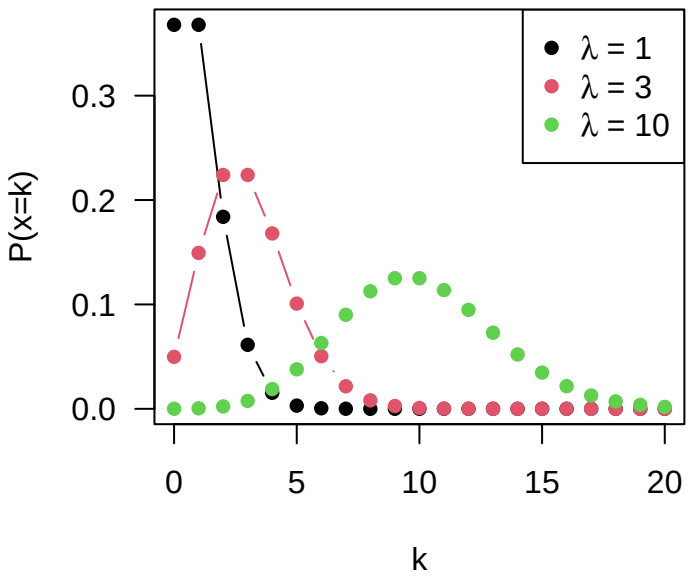
number of offspring. For such data, the data distribution is often skewed, and the variance tends to increase with the mean. The Poisson distribution is tailored for such data.

```
x = rpois(200, 3)
hist(x, las=1)
```



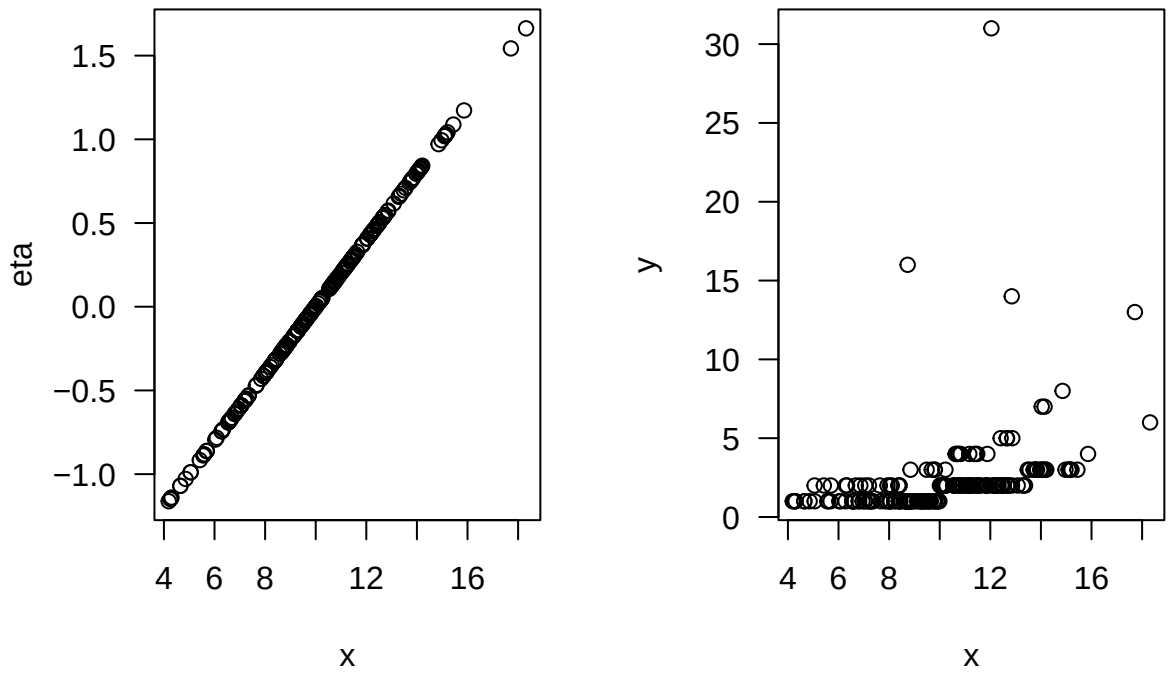
The Poisson distribution has a single parameter λ that determines both the mean and the variance, so that $E(X) = Var(X) = \lambda$. Thus, the variance increases linearly with the mean. The probability mass function changes quite dramatically for different values of λ. For low values the distribution is highly skewed, for high values the distribution approaches a Gaussian (normal) distribution. However, the variance is still constrained to be the same as the mean, which is not the case for the normal distribution.

```
x = seq(0, 20, 1)
y = dpois(x, lambda=1)
plot(x,y, type="b", las=1, xlab="k", ylab="P(x=k)", pch=16, col=1)
points(x, dpois(x, lambda=3), type="b", pch=16, col=2)
points(x, dpois(x, lambda=10), type="b", pch=16, col=3)
legend("topright", col=1:3, pch=16,
       legend=c(expression(paste(lambda, " = 1")),
                expression(paste(lambda, " = 3")),
                expression(paste(lambda, " = 10"))))
```



The distribution of count data can sometimes by normalized through a log-transformation, and the log is indeed the link function of a Poisson regression model. The alternative method of log-transforming the data and then fitting a Gaussian model is problematic when there are zeros in the data. Adding a constant (e.g. 0.5 or 1) is sometimes an option, but is generally not recommended. A better option is to analyze the data in a GLM framework with Poisson-distributed errors and a log link function.

```
x = rnorm(200, 10, 3)
eta = -2 + 0.2*x
y = ceiling(exp(eta + rpois(200, 0.3)))

par(mfrow=c(1,2))
plot(x, eta, las=1)
plot(x, y, las=1)
```

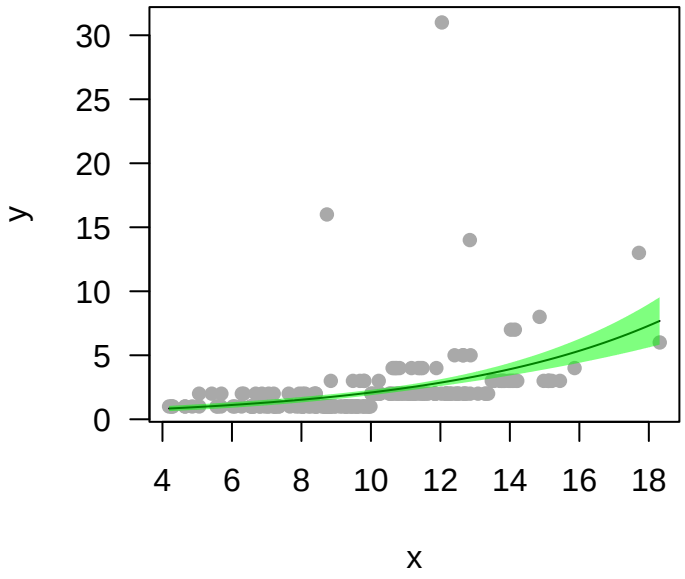



```
m = glm(y~x, family="poisson")
summary(m)
```

```
##
## Call:
## glm(formula = y ~ x, family = "poisson")
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -0.9261  -0.6100  -0.3920   0.0437   9.5447
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept) -0.82795    0.18931  -4.374 1.22e-05 ***
## x            0.15656    0.01624   9.639  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for poisson family taken to be 1)
##
##     Null deviance: 319.56  on 199  degrees of freedom
## Residual deviance: 227.23  on 198  degrees of freedom
## AIC: 737.22
##
## Number of Fisher Scoring iterations: 5
```

When interpreting Poisson regressions, there are several things to keep in mind. First, as in all GLMs, the parameters are reported on the link scale, here log. Recall that the link scale has useful proportional properties, so that the slope can be interpreted roughly as the proportional change in y per unit change in x. To plot the fitted regression line, we have to back-transform the predicted values. This time we use the generic `predict` function to obtain the predicted values on the data scale, and to construct a 95% confidence polygon.

```
plot(x, y, las=1, col="darkgrey", pch=16)
xx = seq(min(x), max(x), 0.01)
y_hat = predict(m, newdata=list(x=xx), type="response", se.fit=T)
lines(xx, y_hat$fit)
#lines(xx, y_hat$fit+1.96*y_hat$se.fit, lty=2)
#lines(xx, y_hat$fit-1.96*y_hat$se.fit, lty=2)

polygon(c(xx, rev(xx)),
        c(y_hat$fit+1.96*y_hat$se.fit,
          rev(y_hat$fit-1.96*y_hat$se.fit)),
        col = rgb(0,1,0,.5), border = FALSE)
```



As in logistic regression the normal r^2 is not valid, and we can use e.g. the `r.squaredGLMM` function to obtain a Pseudo r^2. For a simple GLM an older Pseudo r^2 is

$$1 - \frac{Residual\ deviance}{Null\ deviance}$$

.

```
r.squaredGLMM(m)
```

```
## Warning: the null model is correct only if all variables used by the original
## model remain unchanged.
```

```
##                 R2m       R2c
## delta     0.3108938 0.3108938
## lognormal 0.3517401 0.3517401
## trigamma  0.2659155 0.2659155
```

```
1-(m$deviance/m$null.deviance)
```

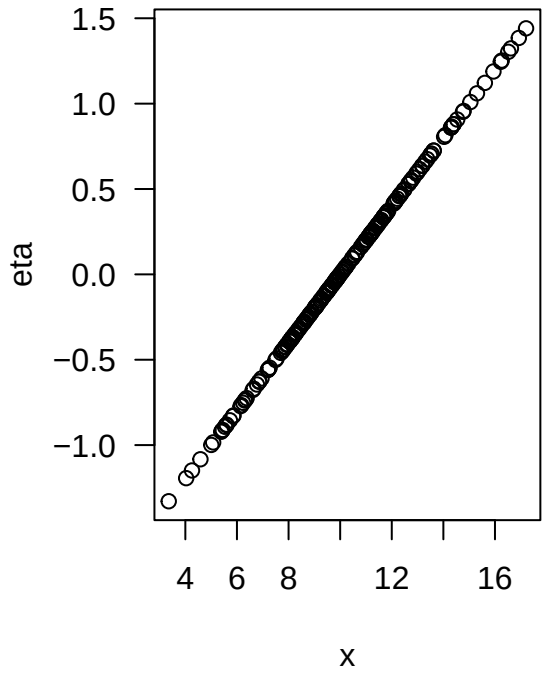
```
## [1] 0.2889314
```

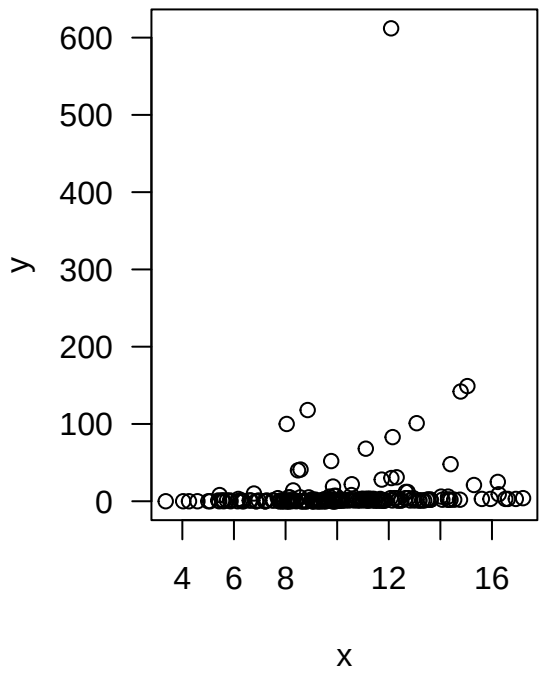
The *deviance* is a measure of how far our model is from a "perfect" or saturated model, i.e. one that perfectly explains the data. Fitting a GLM is done by maximizing the likelihood of a model, which is the same as minimizing the deviance. The deviance is defined mathematically as 2 times the difference in the log likelihood of the model and a saturated model, and can be used to compare alternative models. For example, the Pseudo r^2 above is based on comparing the (mis)fit of the focal model to a null model with only an intercept. If a model is as bad as a null model, the Pseudo r^2 will be 0. We can also use the deviance to compare other models, e.g. through likelihood ratio tests (more on that later).

Second, recall that the Poisson distribution has a single parameter λ determining both the mean and the variance. In real count data the variance often increase disproportionally compared to the mean, a phenomenon called *overdispersion*. Biologically, this occurs because we tend to observe multiple entities together. For example, in a survey of common eiders, we will often see either no eiders, or a lot of eiders.

We can quantify overdispersion in the data based on the fitted model by calculating the ratio of the residual deviance to the residual degrees of freedom. In the model above there is no serious overdispersion, because the residual deviance is only a little larger than the residual degrees of freedom. Let us construct an example with more serious overdispersion.

```
set.seed(1)
x = rnorm(200, 10, 3)
eta = -2 + 0.2*x
y = floor(exp(eta + rnbinom(200, 1, mu=.8)))

par(mfrow=c(1,2))
plot(x, eta, las=1)
plot(x, y, las=1)
```






```
m = glm(y~x, family="poisson")
summary(m)
```

```
##
## Call:
## glm(formula = y ~ x, family = "poisson")
##
## Deviance Residuals:
##    Min      1Q  Median      3Q     Max
## -7.339  -3.851  -3.015  -2.147  59.211
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept) -0.054938   0.094237  -0.583     0.56
## x            0.217419   0.007729  28.129   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for poisson family taken to be 1)
##
##     Null deviance: 8793.6  on 199  degrees of freedom
## Residual deviance: 8005.2  on 198  degrees of freedom
## AIC: 8452.8
##
## Number of Fisher Scoring iterations: 7
```

Here the overdispersion is serious, and we can not trust the model estimates. In this case, we need an alternative link function that allows the variance to increase more than the mean. The negative binomial distribution is a good option. The negative binomial is similar to the Poisson distribution, but includes an additional parameter modelling the disproportionate increase in variance with increasing mean.

```
library(MASS)
m = glm.nb(y~x)
```

```
summary(m)
```

```
##
## Call:
## glm.nb(formula = y ~ x, init.theta = 0.2993347963, link = log)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -1.3934  -1.0868  -0.8307  -0.5080   4.6882
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept) -1.17822    0.50977  -2.311   0.0208 *
## x            0.31932    0.04817   6.628 3.39e-11 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for Negative Binomial(0.2993) family taken to be 1)
##
##     Null deviance: 248.15  on 199  degrees of freedom
## Residual deviance: 215.64  on 198  degrees of freedom
## AIC: 1083.2
##
## Number of Fisher Scoring iterations: 1
##
##
##               Theta:  0.2993
##           Std. Err.:  0.0304
##
##  2 x log-likelihood:  -1077.2360
```

Note especially that the estimated standard error is now much larger.

Data exercise: bee distribution

The following dataset includes the abundance of the bee species *Eulaema nigrita* in the Brazilian Atlantic forest, and a number of potential predictor variables including climate (mean annual temperature and precipitation, temperature and precipitation seasonality) and land use (proportion forest cover and land use heterogeneity defined as the Shannon diversity of local land-use classes). Use a GLM to build a model explaining the distribution patterns of *Eulaema nigrita*. Interpret the results and produce nice tables and figures.

```
dat = read.csv("datasets/Eulaema.csv")
head(dat)
```

```
##   Eulaema_nigrita   method   effort altitude MAT  MAP Tseason Pseason
## 1             492 NetTraps 4.189655        6 235 1073    2036      53
## 2             372    Traps 5.204007       17 230  987    1760      49
## 3             679    Traps 5.204007       17 230  987    1760      49
## 4             600    Traps 5.204007       30 231 1030    1820      51
## 5              28      Net 4.969813        0 259 1693    1074      62
```

```
## 6             535      Net 4.969813       43 255 1697    1061      63
##      forest.    lu_het
## 1 0.04416404 1.0531299
## 2 0.18217054 0.7571063
## 3 0.18217054 0.7571063
## 4 0.01577287 0.8277580
## 5 0.05000000 0.9672604
## 6 0.05000000 0.8711333
```

Example analysis

As an example, we will ask whether the local abundance of *Eulaema nigrita* depends on forest cover, while accounting for variation in yearly rainfall. We could write the analysis methods as follows:

To assess the response of *Eulaema nigrita* to variation in forest cover, we fitted a generalized linear model with negative binomial error distribution (accounting for severe overdispersion) and a log link function. To account for the expected greater abundance of the study species in drier areas, we included mean annual precipitation as a covariate in the model.

```
m = glm.nb(Eulaema_nigrita ~ MAP + forest., data = dat)
summary(m)
```

```
##
## Call:
## glm.nb(formula = Eulaema_nigrita ~ MAP + forest., data = dat,
##     init.theta = 0.7545413278, link = log)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.6661  -1.0239  -0.5326   0.1528   3.7939
##
## Coefficients:
##               Estimate Std. Error z value Pr(>|z|)
## (Intercept)  6.6792933  0.3295954  20.265  < 2e-16 ***
## MAP         -0.0013900  0.0002227  -6.242 4.32e-10 ***
## forest.     -1.3118618  0.3170683  -4.137 3.51e-05 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for Negative Binomial(0.7545) family taken to be 1)
##
##     Null deviance: 271.10  on 177  degrees of freedom
## Residual deviance: 211.01  on 175  degrees of freedom
## AIC: 1843.1
##
## Number of Fisher Scoring iterations: 1
##
##
##               Theta:  0.7545
##           Std. Err.:  0.0735
##
##  2 x log-likelihood:  -1835.0750
```

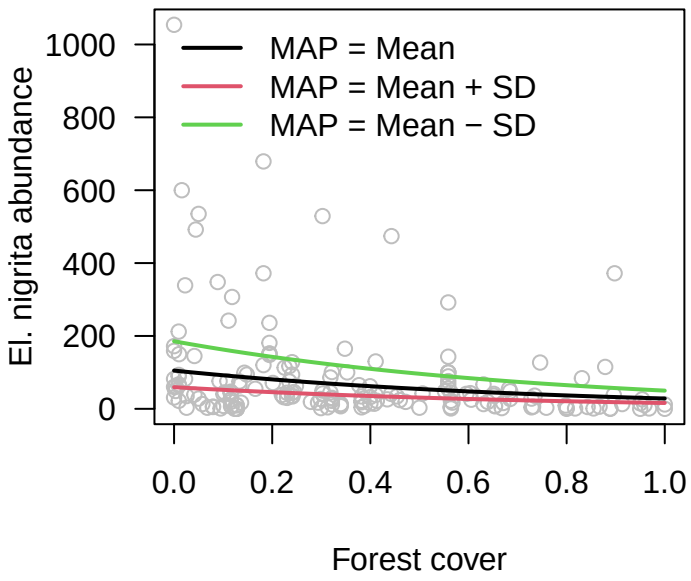
```
1- m$deviance/m$null.deviance
```

```
## [1] 0.2216341
```

Making predictions from multiple-regression models

To illustrate the response of *Eulaema nigrita* to local forest cover, while also representing the effect of rainfall, we plot the response curves for three different fixed values of mean annual precipitation.

```
plot(dat$forest., dat$Eulaema_nigrita, col="grey", las=1,
     xlab="Forest cover",
     ylab="El. nigrita abundance")

newforest = seq(min(dat$forest.), max(dat$forest.), length.out=200)
newMAP = rep(mean(dat$MAP), length(newforest))
y_hat = predict(m, newdata=list(MAP=newMAP,
                                forest.=newforest),
                type="response")
lines(newforest, y_hat,lwd=2)

newMAP2 = rep(mean(dat$MAP)+sd(dat$MAP), length(newforest))
y_hat2 = predict(m, newdata=list(MAP=newMAP2,
                                forest.=newforest),
                type="response")

newMAP3 = rep(mean(dat$MAP)-sd(dat$MAP), length(newforest))
y_hat3 = predict(m, newdata=list(MAP=newMAP3,
                                forest.=newforest),
                type="response")

lines(newforest, y_hat2, lwd=2, col=2)
lines(newforest, y_hat3, lwd=2, col=3)

legend("topleft", lty=1, lwd=2, col=1:3, bty="n",
       legend=c("MAP = Mean",
                "MAP = Mean + SD",
                "MAP = Mean - SD"))
```



Hurdle models

In biology we often have count data with many zeros. This is one of the causes of overdispersion as discussed above. Another way to deal with this is to split the analysis into two independent components, where the first models the zeros, and the second models the counts given that at least 1 entity was observed. This is called a hurdle model, where hurdle refers to the separation of zeros from non-zeros. We can fit a hurdle model in two parts and then combine the predictions.

Below we define two new response variables, where the first (y1) is a 0/1 variable, and the second (y2) has all the zeros set to `NA`.

```
y1 = ((y>1)*1)
m1 = glm(y1~x, family="binomial" (link="logit"))

y2 = y
y2[which(y==0)] = NA

m2 = glm(y2~x, family="poisson", na=na.exclude)
```

```
coefs1 = summary(m1)$coef
coefs2 = summary(m2)$coef
y_hat1 = coefs1[1,1] + coefs1[2,1]*x
y_hat2 = coefs2[1,1] + coefs2[2,1]*x

y_pred = invlogit(y_hat1)*exp(y_hat2)

par(mfrow=c(1,3))
plot(x, invlogit(y_hat1), las=1)
```

```
plot(x, exp(y_hat2), las=1)
plot(x, y_pred, las=1)
```