Quantitative Analysis in Ecology and Evolution

Øystein H. Opedal

5 May 2022

1. Preface

These lectures notes are written for the course BIOS14 at Lund University. They will cover what I consider to be necessary (or at least useful!) knowledge of data analysis and quantitative methods in ecology, evolutionary biology, and related fields. This document is work-in-progress and is not comprehensive. I therefore strongly encourage the reader to consult the many excellent text books available on 'biostatistics', R, quantitative biology etc.

Notes on course literature

Personally I have also found Crawleys brick 'The R Book' useful in the past, it's as much a statistics text as an R text. Another text I like is the book 'Quantitative Ecology and Evolutionary Biology' by Ovaskainen, de Knegt & Delgado.

What separates this document from 'typical' texts in biostatistics is the limited focus on statistical hypothesis testing. Though I will discuss key aspects of hypothesis testing in the traditional sense, focus will be throughout on quantification, parameter estimation, and biological interpretation of results. This choice is made in an attempt to compensate for an issue that occurs frequently in scientific writing, namely that the presentation of results in biological publications has increasingly tended towards a focus on statistical hypothesis testing ('significance testing'), at the cost of focus on the biological significance of any result obtained. An example is sentences on the form 'the treatment significantly ($P < 0.05$) affected the growth of plants' in place of an appropriate sentence such as 'plants in the high-fertilizer treatment grew 40% larger than those in the low-fertilizer treatment'. There is now a growing focus on turning this trend (Wasserstein & Lazar 2016), and this must start with the way statistics are though in biostatistical courses.

Beyond quantification and data analysis, the following chapters will also introduce some basic data handling, scientific programming, and graphics. These skills are by now essential for most practicing researchers, and I like to think the skills will be useful also for those that put their Biology education to use outside of academia.

2. Measurement and meaning in biology

Measurement theory is concerned with the relationship between measurements and the theoretical context in which they will function. In other words, what are the numbers we record meant to represent? The title of this section is shared with an important paper published in 2010 by evolutionary biologist David Houle and colleagues, in which they pointed out the common misconnect in biology between the measurements taken in the field or laboratory, and the biological properties they are meant to represent. This exemplifies a more general problem across biology, where the interpretation of measurements and analyses fails to focus on the biological questions that motivated the study in the first place. Before reading on, I strongly suggest to read at least the first pages of Houle et al. 2010, to get a grasp of the problem at hand.

Scale types and transformations

The concept of scale types has rarely been thought in biology, yet any quantitative measurement is placed on a specific scale type, with some associated properties such as permissable transformations. The perhaps most familiar scale type is the *ratio scale*, represented for example by any linear size measurement (with units such as *mm* or *m*). Importantly, the ratio scale has a natural zero, and negative values don't occur.

In contrast, consider the measurement of days since January 1st (often used to record for example breeding dates of birds). Dates are on an *interval scale*, and lacks a natural zero (i.e., the zero is chosen arbitrarily). This has consequences, for example, for how we compare raw and proportional variances among groups (see below). On this topic, it is common to see the number of days since January 1st labelled as a 'Julian date'. I write this on the 7th of May 2022, and the Julian date is 22127 (the number of days since the beginning of the Julian period).

A third common scale type in biology is the *ordinal scale*, where measurements are in order (e.g. 2 is larger than 1), but the intervals between values are not fixed (i.e. the difference between 1 and 2 may differ from the difference between 2 and 3). Examples include where species are scored within a plot as "dominant", "common", "rare", and "very rare". Most will agree that "common" > "rare", but it does not make sense to translate these categories into numerical values (e.g. 4 for "dominant", 3 for "common" etc.) and then compute the mean.

When measurements are categories (e.g. red vs. blue vs. green), they are on a *nominal scale*.

3. Summary statistics

Before departing on any data analysis, it is advisable to explore the data at hand both graphically, and by obtaining relevant summary statistics. Summary statistics are those that aim to describe the properties of the data. For example, the central tendency of a dataset is generally measured by the arithmetic mean (typically denoted μ or $\bar{x}$), median, or, less frequently, the mode. Which of these to choose depends in part on the distribution of the data. If the data are skewed (tendency towards large or small values), the mean can give a misleading picture of the central tendency. The same issue arises if there are extreme values (outliers) that will tend to affect the mean much more than they affect the median. In some cases, a good option can be to report both the mean and the median, which together will give a more complete impression of the data distribution. For ordinal or nominal variables, the mode may be a suitable option.

Variation in the data is typically described by the variance (typically denoted σ^2) or its square root, the standard deviation ($\sqrt{\sigma^2} = \sigma$). As pointed out by R. A. Fisher, the advantage of the variance (mean squared deviation from the mean) is that variance components are additive, allowing us to partition the total variance in the data into different components (see more on Variance partitioning below). To understand why square values are additive, recall Pythagoras theorem

$c^2 = a^2 + b^2$, but $c \neq a + b$).

The standard deviation measures the mean deviation of each data point from the mean, and thus has the advantage of having the same units as the original measurements. If we have measured a trait in *mm*, the standard deviation gives the average deviation in *mm* from the mean, and is thus easy to interpret.

Note that the variance and standard deviation are measures of dispersion in the data, not the certainty of an estimate. Therefore, unlike *standard errors* ($SE = SD/\sqrt{n}$), the standard deviation should not be given with the $\pm$ sign. This mistake is very frequent in papers in biology.

Very often we are interested in measuring the proportional variation in a variable. The idea is that, because larger things are more variable than small things, we may want to compare how variable different entities are, *proportional to their mean*. One common measure is the coefficient of variation (CV), defined as

$CV = \frac{\sigma}{\mu}$

The CV is often miscalculated in published studies, so keep you eyes open. Because the standard deviation σ is the square root of the variance σ^2, the CV is often written as

$CV = \frac{\sqrt{\sigma^2}}{\mu}$

which is sometimes misinterpreted or even mistyped by the journal typesetter as

$CV = \sqrt{\frac{\sigma^2}{\mu}}$.

Another method for placing data on a proportional scale is taking natural logarithms. The standard deviation of a log-transformed variable is very similar to the standard deviation on arithmetic scale divided by the trait mean (the CV), as long as the variance is much smaller than the mean.

Another way to see the nice proportional properties of the log-scale is to recall that $log(\frac{a}{b}) = -log(\frac{b}{a})$

```
log(1.1/1)
```

```
## [1] 0.09531018
```

```
-log(1/1.1)
```

```
## [1] 0.09531018
```

Note also that when a (1.1) is 10% larger than b (1.0), the *log ratio* is ~0.1. Recall that $log(\frac{a}{b}) = log(a) - log(b)$, and thus differences on log scale multiplied by 100 are roughly interpretable as difference in percent (when a and b are not very different). Log ratios are therefore often good measures of effect size.

Simulating data from statistical distributions

Throughout this course we will use simulated data to illustrate concepts and ideas, and to understand how statistical models work. `R` has built-in functions for simulating data from many statistical distributions, including the normal distribution. The function `rnorm()` takes three main arguments, the number of samples `n`, the mean, and the standard deviation `sd`.

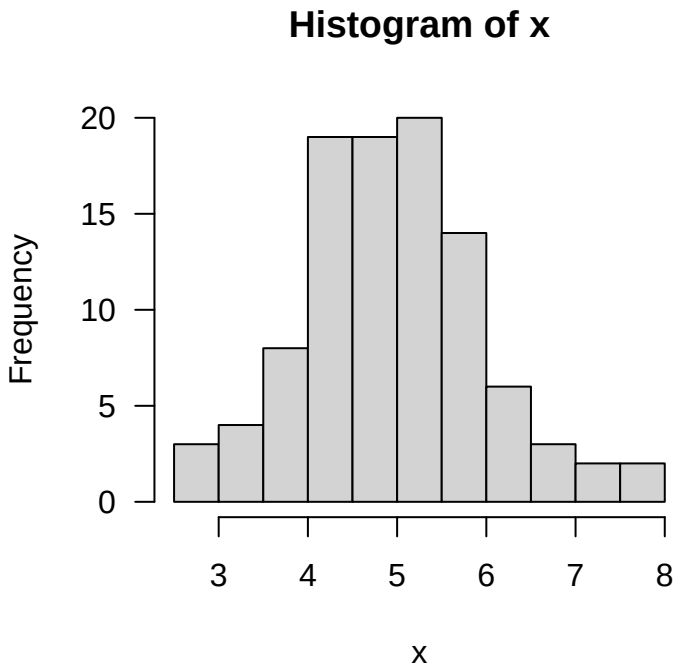
```
x = rnorm(n=100, mean=5, sd=1)
mean(x)
```

```
## [1] 4.960858
```

```
sd(x)
```

```
## [1] 1.015808
```

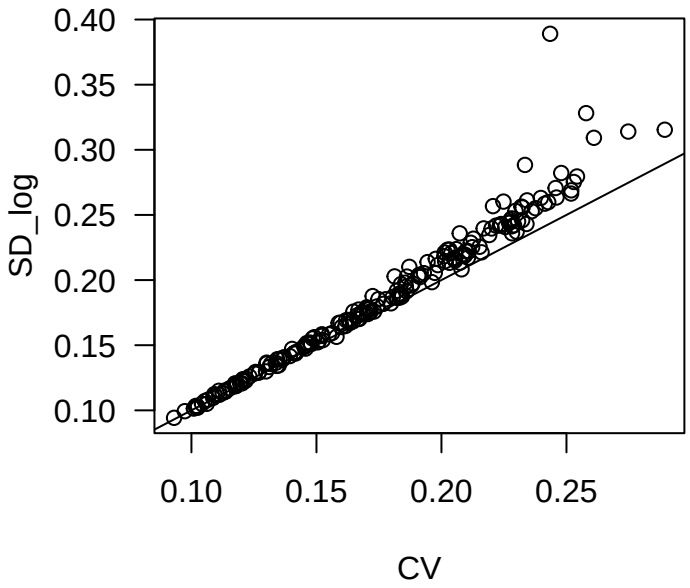
```
hist(x, las=1)
```



R functions know the position of arguments, so that `rnorm(100, 5, 1)` will give the same result as above.

Exercise: The proportional properties of the natural log

Use simulated data to show the close relationship between the SD of log-transformed data and the *CV* on arithmetic scale. You may need e.g. the `rnorm` function and a *for*-loop to achieve this. One strategy would be to start with comparing the two values for a single case, then build a matrix to hold the paired values, and finally use a *for*-loop to populate the matrix. see Appendix 1 for help to get started with programming. The following figure illustrates the kind of pattern we expect.



Bootstrapping

Bootstrapping is a common resampling technique used to assess uncertainty in variables. This can be especially relevant when the variable of interest is already a summary of a statistical population, such as a coefficent of variation (CV). As a first example, we will use bootstrapping to obtain the standard error of the mean, which we have already seen is given by $SE = \sqrt{Var/n}$.

To ensure reproducibility of the results (i.e. that we will get the same result every time, even when we work on different computers), we set the "seed" of the random number generator using the `set.seed()` function.

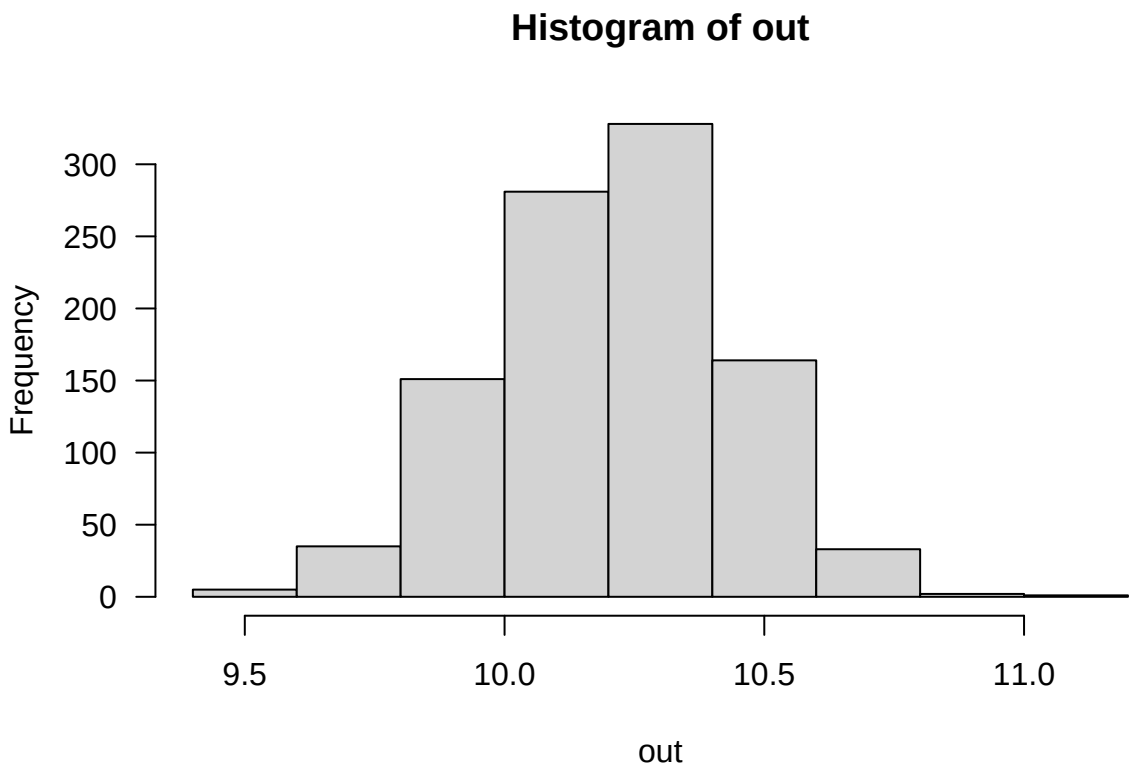
```
set.seed(1)
x = rnorm(50, 10, 2)
se_x = sqrt(var(x)/length(x))
```

We will do a non-parametric bootstrap, where we resample the data many times. For each resampling, we maintain the original sample size (here 50), but we draw from the data with replacement, so that by chance some values will be sampled several times and others not at all.

```
out = NULL
for(i in 1:1000){
  sample = sample(x, replace=TRUE)
  out[i] = mean(sample)
}
```

The variable `out` now contains what we can call the sampling distribution of the mean of x. The standard deviation of the sampling distribution gives an approximation of the standard error.

```
hist(out, las=1)
```




```
sd(out)
```

```
## [1] 0.2307834
```

As expected, this is close to the theoretical standard error

```
se_x
```

```
## [1] 0.2351537
```

Given that we also have the full sampling distribution, we can also choose to derive some quantiles, such as a 95% confidence interval.

```
quantile(out, c(0.025, 0.975))
```

```
##      2.5%     97.5%
##  9.760249 10.624404
```

3. The linear model I: Introduction and linear regression

Nearly all the statistical models we will discuss in this text are forms of the linear model

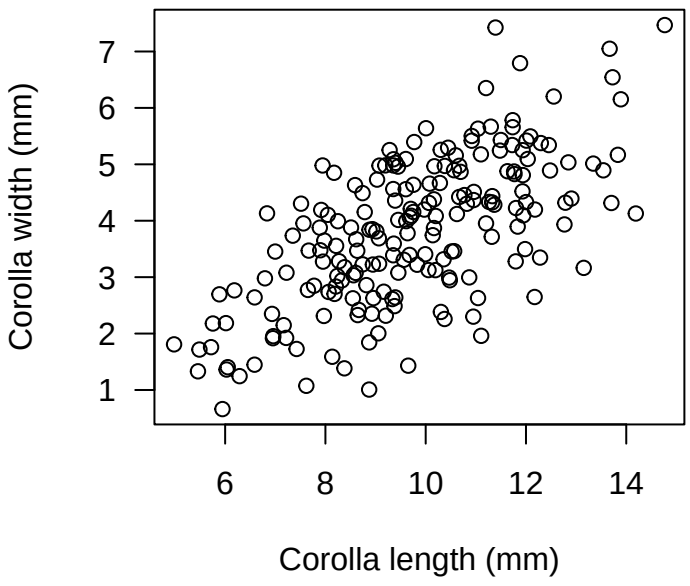
$y_i = \beta_0 + \Sigma_j x_i \beta_j + \epsilon_i$

The term β_0 (sometimes denoted α) is the *intercept*, which in the context of a linear regression gives the value of the response variable y when the predictor variable x is zero. The β_j are the coefficients ('slopes') for the predictor variables x, and the ϵ represent the *residuals*, the deviations of each data point from it's expected value based on the fitted model. In a regression, the residual is the perpendicular distance from the fitted regression line to each data point. The linear model assumes that the residuals (not the data!) are normally distributed, though minor deviations from this is not generally a problem.

In the following, we will consider a series of examples of linear models fitted to simulated data. After simulating some values of the predictor x, we define y as a function of x and add some residual variance to the data (i.e. we simulate data from the same linear model that we will eventually fit to the data.) The advantage of starting from simulated data is that we know the true values of the parameters we will try to estimate. This is very useful when we want to check that our analysis is doing what we think it is doing.

As noted above, `R` functions know the order of arguments. Note how the second incidence of `rnorm` below skips the formalities.

```
set.seed(85)
x = rnorm(n=200, mean=10, sd=2)
y = 0.4*x + rnorm(200, 0, 1)

plot(x, y, las=1,
     xlab="Corolla length (mm)",
     ylab="Corolla width (mm)")
```



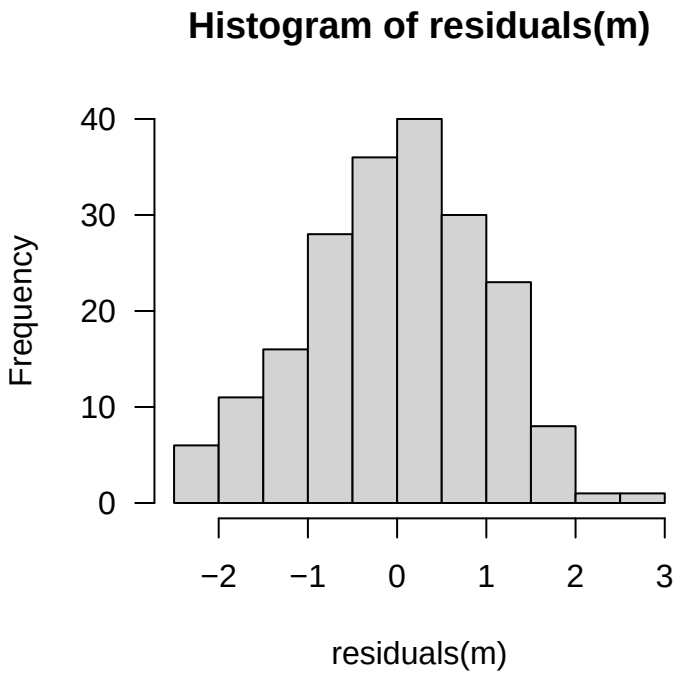
At this point let's talk about how to write `R` code. In the `R` chunks above you may start to notice some 'style conventions'. When we are new to `R` and coding, most of us write really messy code. It is now becoming

mandatory to publish our code alongside our papers, and we thus have to learn to write code that is easy to read and pleasent to look at. I don't follow all the 'rules' around this, but at least I try to be consistent. For example, the strict `R` convention has been to use the `<-` operator for assignments, but I find the `=` easier. Note though the spaces to either side of `=` that makes the code easier to read. Within functions, add a space after commas.

In scatterplots, a useful change from the default is to make the y-axis labels horizontal by setting `las=1`. There are hundreds of ways to change the appearence of `R`-plots, see the wonderful world of `par()`. If you prefer, you can choose to learn alternative plotting frameworks such as `ggplot`', but these lecture notes will use `R` packages only when strictly needed.

The aim of regression analysis is to estimate the linear relationship between a response variable (or 'dependent variable') and one or more predictor variables ('independent variables'). The most common form of regression analysis is so-called ordinary least-square (OLS) regression, in which the regression parameters are estimated so as to minimize the square deviations of the data points from the estimated regression line. The deviations are termed *residuals*, and are assumed to be normally distributed.

```
m = lm(y~x)
hist(residuals(m), las=1)
```



These residuals are fine, as is of course fully expected given that we simulated the data from the (gaussian) linear model. There are many other ways of assessing whether the model assumptions are met, see for example what happens if you call `plot(m)` after setting `par(mfrow=c(2,2))`. Notive that the `plot` function is *generic*, it produces a different result depending on what is fed to it. If we call `plot(x,y)` when both `x` and `y` are continuous variables, we get a scatterplot. If `x` is a factor, we get a boxplot.

Let's now have a look at the results of our linear model fit.

```
summary(m)
```

```
##
```

```
## Call:
## lm(formula = y ~ x)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -2.45122 -0.68319  0.02913  0.69861  2.88937
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) -0.40114    0.35186   -1.14    0.256
## x            0.43330    0.03538   12.25   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.9912 on 198 degrees of freedom
## Multiple R-squared:  0.4311, Adjusted R-squared:  0.4282
## F-statistic:   150 on 1 and 198 DF,  p-value: < 2.2e-16
```

This summary contains a lot of information. First, we can see some quantiles of the residual distribution, which confirms what we have already seen from the histogram: the residuals are fine because the median is close to zero, the 1st and 3rd quartile are symmetrical, and the min and max values are nearly symmetrical too.

Next we see the model parameter estimates, their standard errors, a test statistic (t), and a P-value. It is tempting to look first at the P-value, the magic measure of significance and, to some, 'importance'. Before going on, let's take a moment to recall what the P-value means and how it is obtained. In the context of the linear regression above, the test statistic t is given by $t = \frac{\hat{\beta}}{SE(\beta)}$. We already know that the standard error $SE = \frac{\sigma}{\sqrt{n}}$, thus

$t = \frac{\hat{\beta}}{\frac{\sigma(\hat{\beta})}{\sqrt{n}}}$

The most important thing to notice here is that the sample size n is in the denominator of the expression for the standard error, so that larger sample size will lead to a smaller standard error, and thus a greater t-value.

The P-value is the probability of observing the observed value of the test statistic given that the null hypothesis (here, a slope of zero) is true, or $P_{obs} = Pr(T > t_{obs} = t(X_{obs}|H_0))$. In other words, it represents the probability that we would have obtained our results by chance.

Because the P-value is obtained by comparing our observed test statistic t to its known distribution, and t increases with sample size, it follows that when the sample size increases, anything will at some point be statistically significant. This is the reason why there are now increasing calls for abandoning P-values as the standard measure of statistical significance. That being said P-values do provide a 'quick and dirty' way of assessing statistical support, and can help guide our interpretation of the results. We will later return to alternative methods of evaluating statistical support, but for now we focus on the more important point: interpretation of the results needs to be done in light of the parameter estimates, their units, and their consequences within the context of the analysis/study.

EXERCISE: Use non-parametric bootstrapping to derive a standard error for the slope of the linear regression above. To do so, sample from the data set, fit the model, and save the sampling distribution for the slope of y on x.

```
## [1] 0.03579301
```

Now, let us return to how we interpret the results of our linear regression. The slope of y on x is about 0.43. Although regression slopes are very often reported without any units, it is important to remember that the

slopes in fact carry the units of both the response and predictor variables. In our example the response and predictor are both measured in *mm*, and the slope is therefore 0.43 *mm/mm*. When we report this in the text, we generally want also to report the standard error, i.e. *slope* $= 0.43 \pm 0.04$ *mm/mm*. Thus, in our example, the response variable increases by 0.43 *mm* per *mm* increase in the predictor. The small standard error (relative to the slope estimate) directly indicates the strong statistical support.

To facilitate further interpretation, we can also report the consequences of a realistic change in the predictor variable. Let's say that we want to know how much y changes for a one standard deviation change in x.

```
coefs = summary(m)$coef
(coefs[2,1]*(mean(x) + sd(x))) - (coefs[2,1]*mean(x))
```

```
## [1] 0.8606095
```

Here, we could write in the results section that 'In the study population, corolla width increased by 0.80 *mm* per standard deviation increase in corolla diameter'.

As a special case, a regression where both the response and predictor variable are natural log-transformed will have a slope interpretable as an *elasticity*, which describes the % change in the response per % change in the predictor. THis is another example of the nice proportional prorerties of the natural log.

Another important parameter in the summary table is the coefficient of determination, the r^2. In our simple univariate regression, the r^2 is simply the square of the Pearson correlation coefficient r between the response and predictor.

```
cor(x,y)^2
```

```
## [1] 0.4310643
```

The r^2 of our model is 0.431, which means that 43.1% of the variance in y is explained by x. In general, it is often nice to report the r^2 directly as a percent (i.e. $\times 100$). To understand why the r^2 gives the % variance explained, note that the r^2 can be computed as the variance in the predicted values $\hat{y}$,

$$V(\hat{y}) = X\beta$$

divided by the total variance in the response varible $V(y)$.

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

```
## [1] 0.7406487
```

```
var(y_hat)/var(y)
```

```
## [1] 0.4310643
```

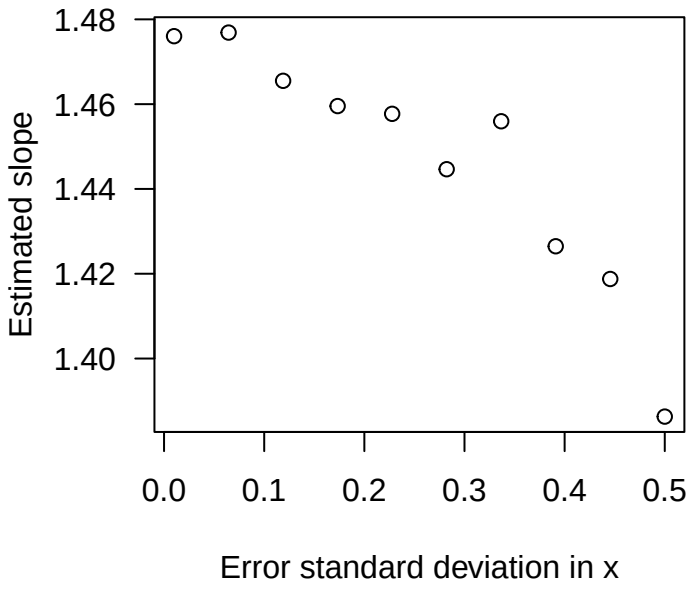
Another way to compute the variance explained by a predictor is $V(x) = \beta_x^2 \sigma_x^2$, where β_x is the parameter estimate (regression slope) for predictor x, and σ_x^2 is the variance of the predictor.

```
coefs[2,1]^2*var(x)
```

```
## [1] 0.7406487
```

Exercise: How error in x- and y-variables affect the slope

The standard linear model assumes that the predictor variable is measured wihtout error. When there is measurement error, this can lead to a bias in the estimated slope. Simulate data with measurement error in the predictor, and produce a plot showing the effect on the estimated slope.



For a simple model like this, the expected attenuation bias (downward bias) in the slope can be estimated by the reliability ratio

$K = 1 - \frac{\sigma_{me}}{\sigma_x}$

where σ_{me} is the measurement error variance and σ_x is the variance in the predictor x. We can thus obtain a corrected slope as

$\beta' = \frac{\beta}{K}$

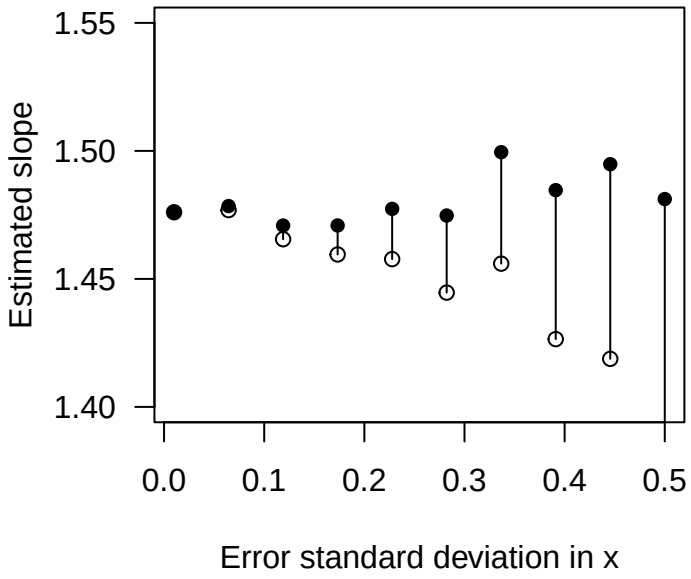
Try to correct your estimated slopes in this way, and a produce a plot showing both the estimated and the corrected slope connected by line segments.

```
relerrors
```

```
##  [1] 2.562152e-05 1.064084e-03 3.621491e-03 7.697843e-03 1.329314e-02
##  [6] 2.040738e-02 2.904057e-02 3.919270e-02 5.086378e-02 6.405380e-02
```

```
corrslope = slope_est/(1-relerrors)

plot(errors, slope_est,
     ylim= c(1.4, 1.55),
     las=1,
     xlab="Error standard deviation in x",
     ylab="Estimated slope")
```

```
points(errors, corrslope, pch=16)
segments(errors, slope_est, errors, corrslope)
```



The linear model II: One-way analysis of variance (ANOVA)

When our predictor variables are categorical (factors), linear models are used to perform analysis of variance. The parameter estimation works in much the same way as in regression, except that instead of estimating regression slopes, we are estimating group effects.

Let's simulate some data, fit a linear model, and perform an ANOVA.

```
groups = as.factor(rep(c("Small", "Medium", "Large"), each=50))
x = c(rnorm(50, 10, 3), rnorm(50, 13, 3), rnorm(50, 14, 3))
m = lm(x~groups)
anova(m)
```

```
## Analysis of Variance Table
##
## Response: x
##            Df  Sum Sq Mean Sq F value    Pr(>F)
## groups      2  433.14 216.568  18.524 6.689e-08 ***
## Residuals 147 1718.59  11.691
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

The ANOVA table contains a lot of information. First, we learn about the number of degrees of freedom for each variable. For the `groups` variable (our focal factor), the 2 degrees of freedom is the number of groups

in our data (3) - 1. The minus 1 comes from the fact that we had to estimate the mean in the data to obtain our sums of squares (the sum of the square deviations of data points from their group means). Similarly for residual degrees of freedom, we have 150 - 2 - 1, where the 2 comes from estimating the two contrasts (difference of group 2 and 3 from group 1), and the 1 is still the estimated mean.

The `Mean Sq` is the variance attributable to each variable, conventionally called the mean sum of squares (the sum of squares divided by the degrees of freedom). The F-ratio is computed as the mean sum of squares for the focal variable divided by the mean residual sum of squares. Thus, it represents the ratio of the among-group variance to the within-group variance, but also the sample size which gives the residual degrees of freedom and thus, larger sample gives lower mean sum of squares and thus higher F-ratios and lower P-values.

In an ANOVA, a statistically significant result such as the one above indicates that at least one group mean is different from the others. To further assess which groups are different, we can extract the typical summary table of the linear model.

```
summary(m)
```

```
##
## Call:
## lm(formula = x ~ groups)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -6.8963 -2.8726 -0.0967  2.3782 10.0157
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept)   13.1284     0.4836  27.150  < 2e-16 ***
## groupsMedium  -0.4296     0.6838  -0.628    0.531
## groupsSmall   -3.8003     0.6838  -5.557 1.25e-07 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 3.419 on 147 degrees of freedom
## Multiple R-squared:  0.2013, Adjusted R-squared:  0.1904
## F-statistic: 18.52 on 2 and 147 DF,  p-value: 6.689e-08
```

This contains some of the same information as the ANOVA table, but we now also obtain parameter estimates. The first parameter, the intercept, corresponds to the estimated mean for the first level of the `groups` factor. In this example this happens to be 'Large', because L comes before M and S in the alphabet. The next two estimates represents *contrasts* from the reference group, and the associated hypothesis tests tests the null hypothesis that the group has the same mean as the reference group.

The parameter estimates allow us to quantify the effect size, i.e. the magnitude of the difference between the groups. A useful way to report such differences is to compute the % difference (the contrast divided by the mean of the reference group, here $3.277/13.4642 = 0.243$), so that we can say that 'Small individuals were 24.3% smaller than large individuals'.

Note that if we want a different reference group, we can change the order of the factor levels.

```
groups = factor(groups, levels=c("Small", "Medium", "Large"))
m = lm(x~groups)
summary(m)
```

```
##
## Call:
## lm(formula = x ~ groups)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -6.8963 -2.8726 -0.0967  2.3782 10.0157
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept)    9.3282     0.4836  19.291  < 2e-16 ***
## groupsMedium   3.3707     0.6838   4.929 2.20e-06 ***
## groupsLarge    3.8003     0.6838   5.557 1.25e-07 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 3.419 on 147 degrees of freedom
## Multiple R-squared:  0.2013, Adjusted R-squared:  0.1904
## F-statistic: 18.52 on 2 and 147 DF,  p-value: 6.689e-08
```

Sometimes we also want to suppress the intercept of the model, and thus estimate the mean and standard error for each level of the predictor. We can do this by adding `-1` to the model formula (what comes after the `~` sign). This could be useful for example, if we wanted to obtain the estimated mean for each group, associated for example with a 95% confidence interval.

```
m = lm(x~groups-1)
summary(m)$coef
```

```
##               Estimate Std. Error  t value     Pr(>|t|)
## groupsSmall   9.328172  0.4835517 19.29095 4.102583e-42
## groupsMedium 12.698877  0.4835517 26.26167 2.232359e-57
## groupsLarge  13.128432  0.4835517 27.15001 3.851221e-59
```

```
confint(m)
```

```
##                  2.5 %   97.5 %
## groupsSmall   8.372561 10.28378
## groupsMedium 11.743266 13.65449
## groupsLarge  12.172821 14.08404
```

Data exercise: Interpreting linear-model analyses

Flowers are integrated phenotypes, which means that the different parts of the flowers are generally covarying with each other so that large flowers have e.g. both longer petals and longer sepals. Evolutionary botanists are interested in these patterns of covariation among floral parts, because they can affect for example the fit of flowers to their pollinators. We will work with a dataset on flower measurements from 9 natural populations in Costa Rica.

The traits are

- ASD: anther-stigma distance (mm)
- GAD: gland-anther distance (mm)

- GSD: gland-stigma distance (mm)
- LBL: lower bract length (mm)
- LBW: lower bract width (mm)
- UBL: upper bract length (mm)
- UBW: upper bract width (mm)
- GW: gland width (mm)
- GA: gland area (mm^2)

The traits have known or assumed functions. Anther-stigma distance is important for the ability of self-pollination, gland-anther distance and gland-stigmas distance affect the fit of flowers to pollinators, the upper and lower bracts are advertisements (think petals in other flowers), and the gland produces the the reward for pollinators.

The first step in any data analysis in always to explore the data. Make a series of histograms and plots. How are the data distributed? Are there any problematic outliers? How are patterns of trait correlations? Which traits are (proportionally) more variable?

What about differences between populations? Are any of the traits detectably different? By 'detectably' I mean statisticaly significant, but because the 's word' is so often misused, I find it safer to say detectably. At the very least, say 'statistically significant', to make clear that you do not (necessarily) imply any biological significance.

To get started, the following lines reads the data.

```
blossoms = read.csv("datasets/blossoms/blossoms.csv")
names(blossoms)
```

```
##  [1] "pop"   "patch" "ASD"   "GAD"   "GSD"   "LBL"   "LBW"   "UBL"   "UBW"
## [10] "GW"    "GA"
```

To summarize the data per population, the `apply` family of functions are useful. To call a function for each level of a factor, such as computing the mean for each population, we can use `tapply`.

```
tapply(blossoms$UBW, blossoms$pop, mean, na.rm=T)
```

```
##       S1      S11      S12       S2      S20      S27       S7       S8
## 17.37067 17.90706 16.82120 19.35714 20.94882 18.64091 18.68200 21.10600
##       S9
## 20.45882
```

A couple of packages are also very useful for producing complete summaries. I use `plyr` and `reshape2`. You could also consider learning some of the more modern things such as `tidyverse`.

```
library(plyr)
library(knitr)
popstats = ddply(blossoms, .(pop), summarize,
                 LBWm = mean(LBW, na.rm=T),
                 LBWsd = sd(LBW, na.rm=T),
                 GSDm = mean(GSD, na.rm=T),
                 GSDsd = sd(GSD, na.rm=T),
                 ASDm = mean(ASD, na.rm=T),
                 ASDsd = sd(ASD, na.rm=T))
popstats[,-1] = round(popstats[,-1], 2)
kable(popstats)
```

pop	LBWm	LBWsd	GSDm	GSDsd	ASDm	ASDsd
S1	18.32	2.13	4.75	0.73	2.56	1.20
S11	18.34	3.68	4.57	0.63	3.16	0.89
S12	17.35	1.34	5.02	0.90	2.66	0.84
S2	20.09	2.62	5.01	0.60	3.87	1.03
S20	21.78	2.58	4.91	0.52	6.32	1.71
S27	19.39	2.09	5.14	0.62	2.98	1.08
S7	19.24	3.76	5.08	0.65	3.92	1.06
S8	20.74	3.10	4.89	0.64	4.52	1.20
S9	20.78	3.68	4.57	0.74	4.05	0.90

After exploring and summarizing the data, fit some linear models to estimate the slopes of one trait on another. Interpret the results. Do the analysis on both arithmetic and log scale. Choose traits that belong to the same vs. different functional groups, can you detect any patterns? Produce tidy figures that illustrate the results.

The linear model III: two-way ANOVA

Analyses of variance can also be performed with more than one factor variable. If we have two factors, we can talk about two-way ANOVA, and so on. A typical example from biology is when we have performed a factorial experiment, and want to assess the effects of each experimental factor and their potential interaction.

With two factors, a full model can be formulated as `y ~ factor1 * factor2`. Recall that in `R`syntax, the * means both main effects and their interaction, while a : means only the interaction. A detectable interaction term in this model would indicate that the effect of factor 1 depends on the level of factor 2 (and *vice versa*). If we are analysing an experiment where we have manipulated both temperature and nitrogen supply, an interaction would mean that the effect of temperature depend on the nitrogen level.

Data exercise: factorial experiment

6. The linear model IV: multiple regression

Linear models are easily extendable to multiple predictor variables. If there are several continuous predictors, the analysis is called a multiple-regression analysis. Multiple regression has some very useful properties. For example, the parameter estimates represent the *marginal effect* of each predcitor, that is the effect of the predictor when all other variables in the model are held constant at their mean. This allows us to evaluate the independent effects of several, potnetially correlated, variables (asking for example which have the stronger effect on the response variable), or to 'control for' some nuisanse variables (say, sampling effort).

```
set.seed(187)
x1 = rnorm(200, 10, 2)
x2 = 0.2*x1 + rnorm(200, 0, 4)
y = 0.7*x1 + 2.2*x2 + rnorm(200, 0, 2)

m = lm(y~x1+x2)

summary(m)
```

```
##
## Call:
## lm(formula = y ~ x1 + x2)
```

```
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -4.2138 -1.3620 -0.0033  1.3520  4.8790
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  0.24361    0.67372   0.362    0.718
## x1           0.66856    0.06499  10.287   <2e-16 ***
## x2           2.19223    0.03211  68.277   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1.809 on 197 degrees of freedom
## Multiple R-squared:  0.9611, Adjusted R-squared:  0.9607
## F-statistic:  2430 on 2 and 197 DF,  p-value: < 2.2e-16
```

```
coefs = summary(m)$coef
```

First, note that the r^2 of the model is 0.961, which means that 96.1% of the variance in y is explained. As before, we can see why this is the case by computing the variance in the predicted values $\hat{y}$,

$$V(\hat{y}) = X\beta$$

, and then divide this by the total variance in the response varible $V(y)$.

```
y_hat = coefs[1,1] + coefs[2,1]*x1 + coefs[3,1]*x2
var(y_hat)
```

```
## [1] 79.93572
```

```
var(y_hat)/var(y)
```

```
## [1] 0.9610509
```

This is the total variance explained by the model. Now what about the variance explained by each of the predictors x_1 and x_2? To compute the predicted values associated only with x_1, we keep x_2 constant at its mean, and *vice versa* for the variance associated with x_2.

```
y_hat1 = coefs[1,1] + coefs[2,1]*x1 + coefs[3,1]*mean(x2)
var(y_hat1)
```

```
## [1] 1.74571
```

```
var(y_hat1)/var(y)
```

```
## [1] 0.02098832
```

```
y_hat2 = coefs[1,1] + coefs[2,1]*mean(x1) + coefs[3,1]*x2
var(y_hat2)
```

```
## [1] 76.89911
```

```
var(y_hat2)/var(y)
```

```
## [1] 0.9245424
```

```
var(y_hat)
```

```
## [1] 79.93572
```

```
var(y_hat1) + var(y_hat2)
```

```
## [1] 78.64482
```

So, what happened to the last few percent of the variance? Recall that

$Var(x + y) = Var(x) + Var(y) + 2Cov(x, y)$.

```
var(y_hat1) + var(y_hat2) + 2*cov(y_hat1, y_hat2)
```

```
## [1] 79.93572
```

As before, we can also do this by computing $V(x) = \beta_x^2 \sigma_x^2$.

```
coefs[2,1]^2*var(x1)
```

```
## [1] 1.74571
```

To include the covariance between the predictors, we can do this in matrix notation $V(\hat{y}) = \hat{\beta}^{\mathbf{T}} \mathbf{S} \hat{\beta}$. Recall the matrix multiplication operator `%*%`.

```
t(coefs[2:3,1]) %*% cov(cbind(x1,x2)) %*% coefs[2:3,1]
```

```
##          [,1]
## [1,] 79.93572
```

The latter approach is the most general, as it extends to any number of predictors in the model. Note that we could, for example, compute the variance explained by a subset of the predictors by specifying the correct vector of β coefficients and their corresponding variance-covariance matrix. This is useful if we had, say, 3 variables related to climate and 3 other variables related to local land-use, and wanted to know how these sets each explain variance in some variable (say, the size of pine trees).

This procedure also hints at a method for obtaining parameter estimates that directly reflect the strength of the effects of each predictor. If all variables had the same variance, then the variance explained would be directly proportional to the regression slope. The most common way to standardize predictor variables is to scale them to zero mean and unit variance, a so-called z-transform

$z = \frac{x - \bar{x}}{\sigma(x)}$

The resulting variable will have a mean of zero and a standard deviation (and variance) or one.

```
x1_z = (x1 - mean(x1))/sd(x1)
x2_z = (x2 - mean(x2))/sd(x2)

m = lm(y~x1_z + x2_z)
summary(m)
```

```
##
## Call:
## lm(formula = y ~ x1_z + x2_z)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -4.2138 -1.3620 -0.0033  1.3520  4.8790
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  12.7948     0.1279  100.03   <2e-16 ***
## x1_z          1.3213     0.1284   10.29   <2e-16 ***
## x2_z          8.7692     0.1284   68.28   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1.809 on 197 degrees of freedom
## Multiple R-squared:  0.9611, Adjusted R-squared:  0.9607
## F-statistic:  2430 on 2 and 197 DF,  p-value: < 2.2e-16
```

Note that the model fit (e.g. the r^2) has not changed, but the parameter estimates have. First, the intercept can now be interpreted as the mean of y, because it represents the value of y when both predictors have a value of 0 (i.e. their mean after the z-transform). This effect can be obtained also by mean-centering the variables without scaling them to a standard deviation of 1.

Second, the slopes now have units of standard deviations, i.e. they describe the change in y per standard deviation change in each predictor. This shown directly that the predictor x_2 explains more variance in y than does x_1.

Another useful trasnformation could be a natural log-transform, or similarly mean-scaling, which would give the slopes units of means, and allow interpreting the change in y per percent change in x.

```
x1_m = (x1 - mean(x1))/mean(x1)
x2_m = (x2 - mean(x2))/mean(x2)

summary(lm(y~x1_m + x2_m))
```

```
##
## Call:
## lm(formula = y ~ x1_m + x2_m)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -4.2138 -1.3620 -0.0033  1.3520  4.8790
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
```

```
## (Intercept) 12.79483    0.12792  100.03   <2e-16 ***
## x1_m         6.79770    0.66079   10.29   <2e-16 ***
## x2_m         5.75352    0.08427   68.28   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1.809 on 197 degrees of freedom
## Multiple R-squared:  0.9611, Adjusted R-squared:  0.9607
## F-statistic:  2430 on 2 and 197 DF,  p-value: < 2.2e-16
```

Multicollinearity

When we have several predictors that are strongly correlated with each other, it becomes difficult to estimate their independent effects. A rule of thumb is that such *multicollinearity* becomes a potential problem when the correlation between the predictors is greater than 0.6 or 0.7. One way of assessing the degree of multicollinearity is to compute *variance inflation factors*, defined as

$VIF_i = \frac{1}{1-r_i^2}$

where the r^2 is from a regression of covariate i on the other covariates included in the model. For our example model, the variance inflation factor for covariate x_1 is thus

```
m1 = lm(x1~x2)
r2 = summary(m1)$r.squared
1/(1-r2)
```

```
## [1] 1.003113
```

This is very low, because the two predictors are not strongly correlated. Rules of thumb for what constitutes severe variance inflation range from $VIF > 3$ to $VIF > 10$. When this occurs, the parameter estimates becomes associated with excessive variance and are thus less reliable. In these cases it may be good to simplify the model by removing some of the correlated predictors, especially if there are several predictors that essentially represent the same property (e.g. multiple measures of body size). If the effects of the correlated predictors are of specific interest, it can also make sense to fit alternative models including each of the candidate predictor, and compare estimates. If the 'best model' is decired, the choice among the predictor may be based on model selection techniques.

The linear model V: Analysis of Covariance (ANCOVA)

In our last section on (simple) linear models, we will consider the case there the predictor variables include both continuous variables and factors. Such analysis are referred to as analyses of covariance.

An ANCOVA-analysis can be used to estimate and test for differences in slopes and intercepts between groups. This comes up often in ecology and evolution, because we often want e.g. to compare the slope of a regression among groups such as sexes, experimental treatments, or populations.

In a simple case with one factor and one covariate, a full model including both main effects and their interaction (i.e. `y ~ factor * covariate` allows differences in both slopes and intercepts between the factor levels, while a simpler model without the interaction term (`y ~ factor + covariate`) allows only differences in intercepts.

DATA EXERCISE

7. Generalized linear models I: Introduction

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

$\eta = \beta_0 + \Sigma_j x_i \beta_j + \epsilon_i$ 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 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 Bernouli 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.

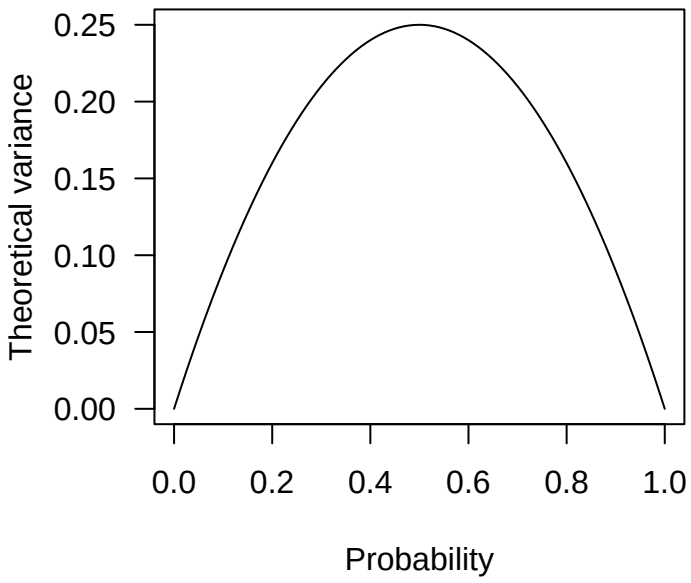
The theoretical variance of the binomial distribution is given by

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

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

```
## [1] 1 7 8
```

```
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 varaince, 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,

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

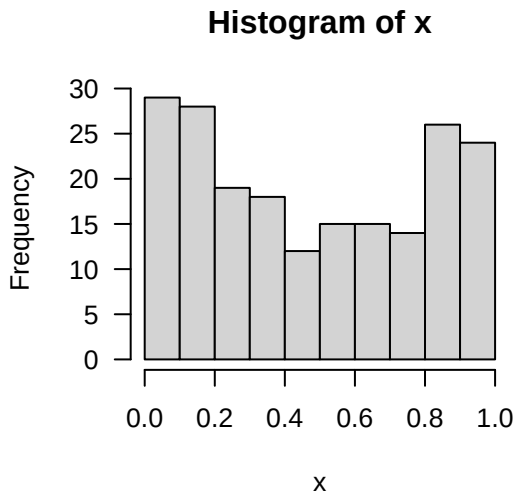
A more meaningful transformation is the logit or log odds transformation

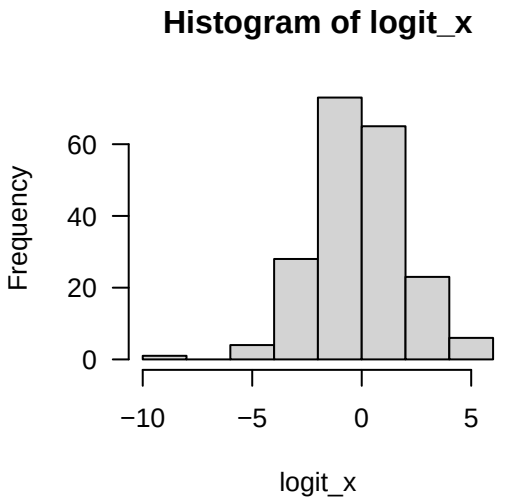
$logit(x) = log(\frac{x}{1-x})$

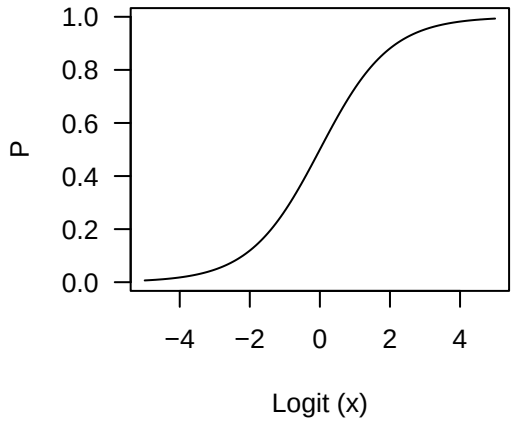
In the following example we are using some functions. Note that when functions are given on a single line, we can skip the special curvy brackets ({).

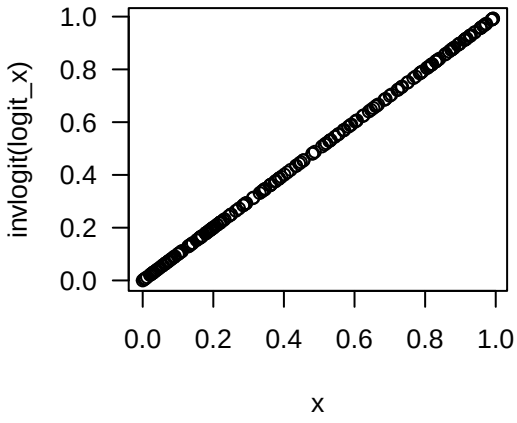
```
logit = function(x) log(x/(1-x))
invlogit = function(x) 1/(1+exp(-x))

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)
```





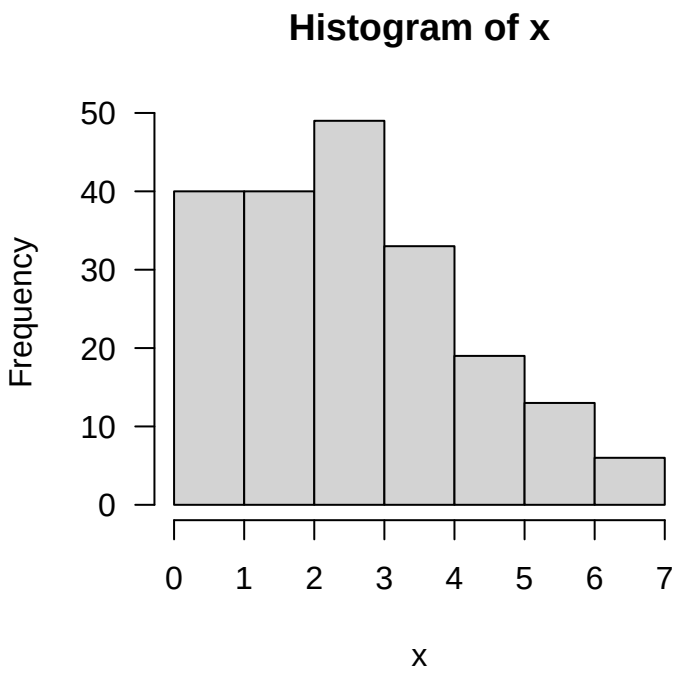




The logit transformation is the most common *link function* in a Generalized Linear Model with binomial errors.

A 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)
```



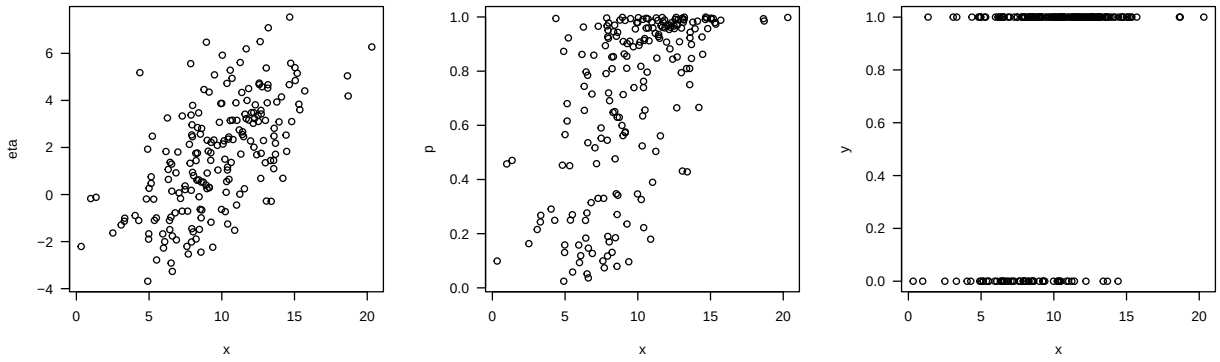
The Poisson distribution has a single parameter λ that determines both the mean and the variance. Thus, the variance increases linearly with the mean.

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 altenrative 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.

7. Generalized linear models II: 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
## -2.2198  -1.0639   0.5421   0.8528   1.8277
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept) -1.85967    0.54217  -3.430 0.000604 ***
## x            0.29369    0.05915   4.965 6.87e-07 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 246.02  on 199  degrees of freedom
## Residual deviance: 214.73  on 198  degrees of freedom
## AIC: 218.73
##
## 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.

$0 = \beta_0 + \beta_x$

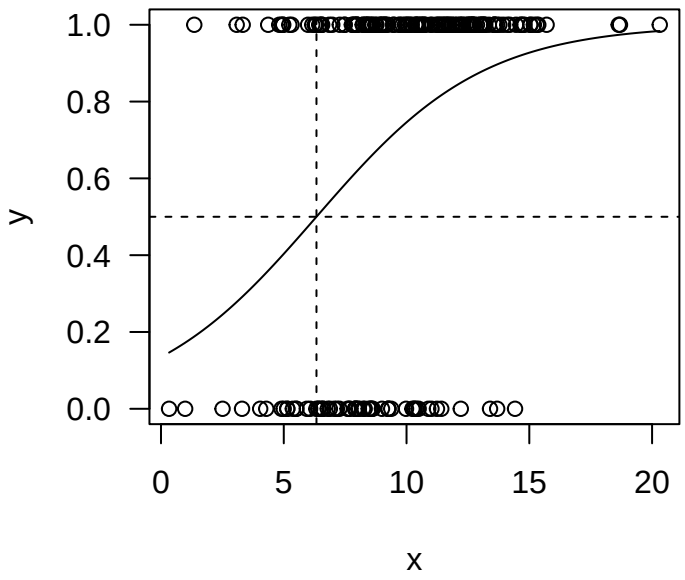
$\frac{-\beta_0}{\beta_x} = x$

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

```
## [1] -0.3643758
```

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 $\hat{p}$. Below we also add some lines to show that our calculations for the predcitor value corresponding to a probability of 0.5 was correct.

```
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)

plot(x, y, las=1)
lines(x_pred, p_hat)
abline(h=0.5, lty=2)
abline(v=-coefs[1,1]/coefs[2,1], lty=2)
```



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 0.2258883 0.2258883
## delta       0.1690865 0.1690865
```

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] 0.1473414
```

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?

8. Generalized linear models II: Poisson and negative-binomial regression

Overdisperson

Negative binomial

8. Generalized linear models III: Hurdle models

8. Mixed-effect models I: Introduction

One very common extension of the linear model is the linear mixed model. The 'mixed' comes from the fact that these models include two variable types: fixed effects and random effects.

Model equation

The fixed effects are the standard predictor variables of a linear model, i.e. variables for which we are interested in their (independent) effect on the response variable. For example, in an ANOVA-type analysis of a factorial experiments, the experimental factors will be treated as fixed effects.

The random effects are variables for which we are not necessarily interested in the mean value of the response for each value of the predictor, but rather the variance in these effects. A common use of random effects is to account for the non-independence of observations that arise, for example, when several measurements are taken from the same individual. Failing to account for this would lead to an artificial inflation of the degrees of freedom of the analysis. This issue is called *pseudoreplication*, because it uses non-independent data points as replicates.

Beyond modelling patterns of non-independence in the data, random-effect models are also often use to estimate variance components that may of direct interest. A typical applicaiton is in quantitative genetics, where the aim of a study can be to estimate the components of the variance in a phenotypic trait. A simple model can be

$y_i = g_i + e_i$

where g is the genetic variance component ande e is the environmental variance component. Estimating these variance components examplifies the general approach of *variance component analysis*.

Variance component analysis using random-effect models

Random-effect models allow us to estimate the variance residing at multiple levels, and thus to ask for example what percentage of variation in a variable is due to differences among populations, and to differences among individuals within populations.

Consider the following simulated data.

```
popmeans = rnorm(10, 20, 4)
```

To specify a random-effect model with the `glmmTMB` package, we use the `(1|pop` format.

```
#library(lme4)
#m = lmer(z~1+(1|pop), data=data)
```

Data exercise: Variance partitioning with random-effects models.

Pick any of the datasets we have worked with in the course that includes at least one grouping variable, and perform a random-effect variance paritioning. Produce a neat table and interpret the results biologically and statistically.

Model selection in confirmatory vs. exploratory analyses

As we have seen in many of examples above, statistical modelling is often used to estimate the parameters of a predefined model representing an theoretically expected relationship, or a biological hypothesis. In these cases, 'model selection' is done by the researcher before performing the analysis, and the model structure is kept fixed whatever the parameter estimates and their uncertainty may be (they are in any case the results to be reported). Such analyses can be seen as 'confirmatory', i.e. they are used to confirm the patterns in the data under the preselected statistical (and corresponding biological) model.

Confirmatory analyses can however involve modification of model structure. A typical example is ANCOVA-type analyses, where one typically starts form a full model allowing differences in both slopes and intercepts, and then simplify the model by dropping first the interaction term and eventually linear terms.

In more complex analyses with several to many candidate predictors, the model structure is not well defines beforehand, and the analyses can be seen as 'exploratory'. Traditional statistical textbooks give detailed account of stretegies for model selection of this kind, such as 'backward selection' with the aim of reducing the model to a 'minimum adequate model', where all terms are statistically significant (referred to as the principle of parsimony, Occam's razor etc.).

Information criteria

Due to the many criticisms of P-values, information criteria have emerged as an alternative approach for selecting among competing models. The philosophy behind these is to maximize the 'information' carried by a model (or, strictly, minimizing the information lost), under the constraint of keeping the model as simple as possible. Thus, they are typically based on comparing the log likelihood of the alternative, nested models penalized by the number of parameters in each model. Nested models means that one model is a special case of the other, e.g. that a certain parameter of the more complex model is set to zero.

Because the AIC value is directly based on the (log) likelihood, it is important that the candidate models are fitted to the same data. Thus, if we have missing values for some predictors, we should remove those observations completely before fitting the the candidate models.

The most common information criterion is the AIC, the 'Akaike Information Criterion' defined as $AIC = -2ln(\hat{L}) + 2k$, where $ln(\hat{L}$ is the log likelihood of the model and k is the number of free parameters in the model. Recall that the likelihood represents the probability of the data given some parameters, and is what is maximized when we fit models with maximum likelihood). The lower the AIC value, the better the model.

$AICc = AIC + \frac{2k(k+1)}{n-k-1}$

The AIC values of several models are often summarized as differences, ΔAIC, from the highest ranked model. Another way to quantify the relative performance of a set of candidate model is to compute weights as

$w_i = \frac{exp(-\frac{1}{2}\Delta_i)}{\sum_{r=1}^{R} exp(-\frac{1}{2}\Delta_r)}$

where the Δ values are the ΔAIC.

As an example, we simulate some data with two candidate predictors. Given a set of nested candidate models, we can build the AIC table as follows

```
set.seed(12)
x1 = rnorm(200, 10, 3)
group = as.factor(sample(c("A", "B"), 200, replace=T))
y = 0.5*x1 + rnorm(200, 0, 4)
y[group=="A"] = y[group=="A"] + rnorm(length(y[group=="A"]), 2, 1)

m1 = lm(y ~ x1 * group)
m2 = lm(y ~ x1 + group)
m3 = lm(y ~ x1)
m4 = lm(y ~ group)
m5 = lm(y ~ 1)

mlist = list(m1, m2, m3, m4, m5)
AICTab = AIC(m1, m2, m3, m4, m5)
AICTab$logLik = unlist(lapply(mlist, logLik))
AICTab = AICTab[order(AICTab$AIC, decreasing=F),]
```

```
AICTab$delta = round(AICTab$AIC - min(AICTab$AIC), 2)
lh = exp(-0.5*AICTab$delta)
AICTab$w = round(lh/sum(lh), 2)
AICTab
```

```
##    df      AIC    logLik delta    w
## m2  4 1110.687 -551.3437  0.00 0.69
## m1  5 1112.549 -551.2745  1.86 0.27
## m3  3 1117.746 -555.8731  7.06 0.02
## m4  3 1119.242 -556.6211  8.55 0.01
## m5  2 1124.156 -560.0779 13.47 0.00
```

Data exercise: Model selection

Pick any of the datasets we have worked with in this course that includes more than one candidate predictor variable. Use AIC to perform model selection, produce a neat summary table with an informative legend, and interpret the results (biologically and statistically).

9. Bayesian methods

All the modelling methods we have discussed so far are fitted by maximimum likelihood methods. This is not the only method for fitting models to data though. In this section we will discuss the philosophy of Bayesian statistics, and some examples of how models can be fitted using Bayesian inference.

Put simply, while hypothesis testing by 'frequentists' is based on estimating the most likely value of parameters, their uncertainty, and P-values, the Bayesian philosophy is explicitly to consider the distribution of plausible parameter values. Furthermore, Bayesian inference involves the formulation of a so-called *prior distribution* that describes our prior belief about which parameter estimates are likely to occur.

At the core of Bayesian inference sit's the simple Bayes theorem or Bayes rule, which states

$P(A|B) = \frac{P(B|A)P(A)}{P(B)}$

Here A and B are events, and the $P(A)$ and $P(B)$ representes the prior belief about the events.

In the context of model fitting, we replace the events A and B with the model parameters θ and the observed data S.

$P(\theta|S) = \frac{P(S|\theta)P(\theta)}{\int P(S|\theta)P(\theta)d\theta}$

where $P(\theta)$ is the prior for the parameters and $P(S|\theta)$ is the likelihood function.

Bayesian model fitting occurs in an iterative process, normally the Monte Carlo Markov Chain (MCMC). A simple method for model fitting by MCMC is the Metropolis-Hastings updating. Briefly, during each iteration a small modification is done to the current value of a parameter, and if this value increases the likelihood of the model (after taking into account the prior for the parameter), the suggestion is accepted and the chain makes the next suggestion based on this updated value. If the suggestion does not improve the likelihood, it is discarded. Through this iterative process, the chain eventually reaches a stable distribution of plausible parameter values, the *posterior distribution*. For complex models this method is generally not feasible, but more sophosticated updating procedures are used that essentially performs the same task (e.g. Gibbs sampling).

The parameter estimates and their uncertainty can be summarized for example as the posterior mean and a credible interval around the mean. Credible intervals in Bayesian analysis corresponds to confidence intervals in maximum-likelihood analses, but are termed differently due to the difference in how they are obtained.

Instead of computing summaries based on the posterior distribution, we can perform downstream analyses for each posterior sample, thus carrying the uncertainty forward and subsequently obtain e.g. posterior means and credible intervals after all analysis steps are complete.

Though Bayesian inference differs philosophically and technically from maximum-likelihood analysis, many applications are ultimately rather similar. For example, we can use Bayesian inference to fit linear models using (nearly) non-informative priors, and obtain nearly identical parameter estimates to those obtained by the `lm` function. Statistical support can be evaluated as e.g. the posterior support, i.e. the proportion of posterior samples that are greater than zero.

Some 'purists' consider themself strictly 'frequentists' or strictly 'Bayesians', but most (the author included) are more than happy to leverage the strengths of specific methods in specific situations.

Fitting a generalized linear mixed model with Bayesian inference

To demonstrate the Bayesian model-fitting procedure, we will use the `Hmsc` package. `Hmsc` stands for Hiererachical Modelling of Species Communities, and implements a modelling framework developed primarily for analyses of multivariate community data. In a later section we will explore multivariate versions of `Hmsc`, but here we will use the package to fit a more standard mixed model. Another option for fitting such models is the `MCMCglmm` package.

The following examples are similar to those given in a vignette associated with the `Hmsc`package, which can be obtained by calling `vignette("vignette_1_univariate", package="Hmsc")`.

As a first very simple example, we will fit a simple linear regression to simulated data. While the `lm` function constructs and fits the model in one go, the `Hmsc`package first constructs the model, and then performs model fitting (posterior sampling) in a second step. This is becasue posterior sampling can take a long time for complex data, and we want to be able to leave it to run e.g. overnight. For the current model though, model fitting is very quick.

```
library(Hmsc)
```

```
## Loading required package: coda
```

```
x = rnorm(200, 10, 3)
y = -2 + 0.4*x + rnorm(200, 0, 2)

m1 = lm(y~x)
m2 = Hmsc(Y = as.matrix(y), XData = data.frame(x), XFormula = ~x,
          distr="normal")

m2 = sampleMcmc(m2, samples=1000, transient=1000, thin=1, verbose=F)
```

```
## setting updater$GammaEta=FALSE due to absence of random effects included to the model
```

```
summary(m1)$coef
```

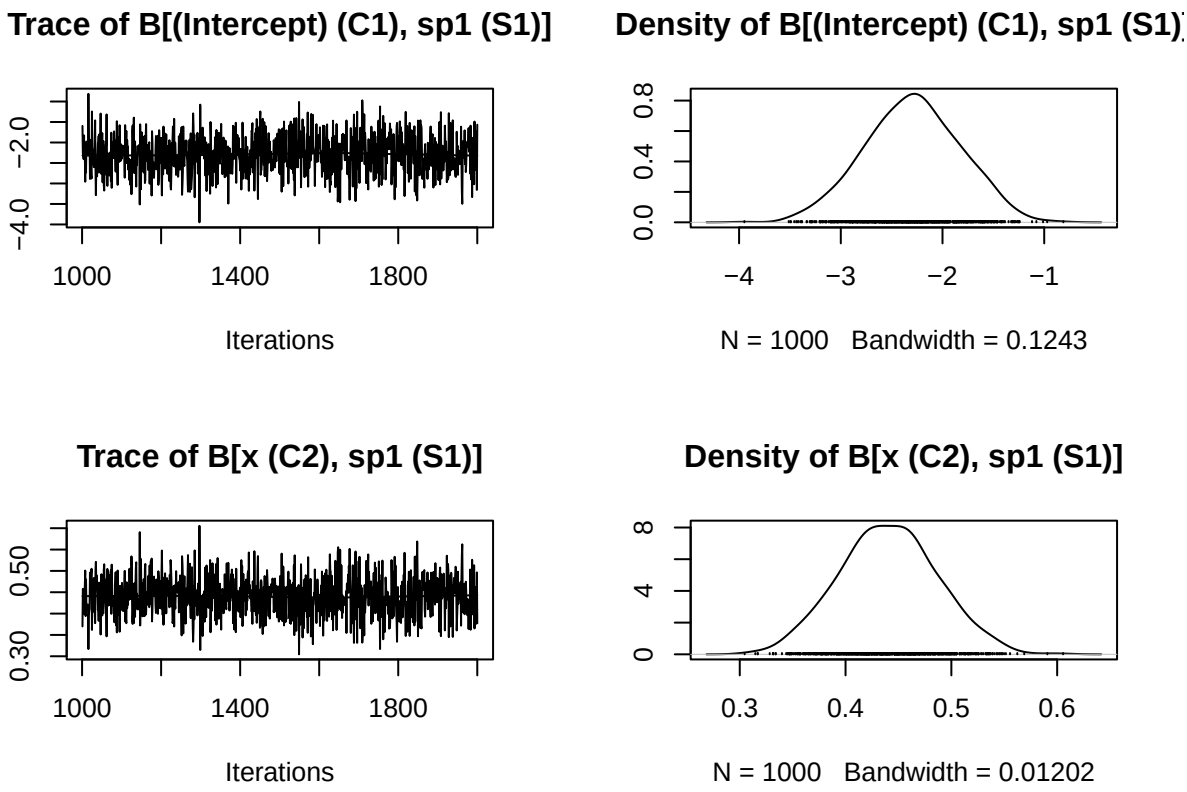
```
##               Estimate Std. Error   t value     Pr(>|t|)
## (Intercept) -2.3326455 0.47652366 -4.895131 2.033509e-06
## x            0.4467981 0.04640133  9.628994 3.040169e-18
```

```
mpost = convertToCodaObject(m2)
summary(mpost$Beta)
```

```
##
## Iterations = 1001:2000
## Thinning interval = 1
## Number of chains = 1
## Sample size per chain = 1000
##
## 1. Empirical mean and standard deviation for each variable,
##    plus standard error of the mean:
##
##                                    Mean     SD Naive SE Time-series SE
## B[(Intercept) (C1), sp1 (S1)] -2.2938 0.4674 0.014780       0.014780
## B[x (C2), sp1 (S1)]            0.4405 0.0462 0.001461       0.001461
##
## 2. Quantiles for each variable:
##
##                                    2.5%     25%     50%     75%   97.5%
## B[(Intercept) (C1), sp1 (S1)] -3.2043 -2.6020 -2.2902 -1.9764 -1.4209
## B[x (C2), sp1 (S1)]            0.3509  0.4097  0.4397  0.4702  0.5334
```

The parameter estimates are very similar, though not identical. This is due to the stochasticity of the MCMC algorithm. The fact that we have sampled the posterior distribution with MCMC also means that, before we start looking more in detail at the model estimates, we should assess whether the model actually converged on a stable solution. One way to do this is to produce a posterior trace plot, which shows how the parameter estimates have changed during the MCMC chain.

```
plot(mpost$Beta)
```



10. Multivariate stats, multivariate mixed models

11. Causal inference, path analysis and structural equation modelling

12. Joint species distribution modelling

HMSC

Other topics to cover somewhere

Parametric vs. non-parametric methods Multicollinearity and VIF

Appendix

Working in R: files, folders and projects

We can make our life easier by learning right from the start how to keep a tidy workflow in R. I recommend learning about `R` 'projects', which become essential once you are working on many parallel tasks/projects. FOr example, I have an `R` project for these course notes. A project is associated to a folder that holds all the project files, including data files, R-code files (analyses, functions), saved results (e.g. model outputs), and saved figures.

One advantage of working within a project is that we don't need to worry about our working directories, it will be automatically set to the project folder. If you also develop a standard naming convention for folders,

if will be easy to remember for example that the input data are kept in the folder 'data', and the figures are saved in the folder 'figures'. This is easier that keeping the files in folders like `C:/Users/Øystein/My documents/stuff/Studies/Lund/Coursework/Master/Stats/R` (I have seen even worse!).

To start a project, simply click `file/New Project`. You will see that there is an option to directly associate the proejct to a system for version control, such as GitHub (see below).

Another common 'mistake' is to do way too many things within one R file. When working with 'real' analyses, it is good style to keep for example one R file for formatting the data (which you of course will not do in Excel!) and writing clean files for the analyses, another for performing e.g. model fitting, and a third to produce graphics based on the results. For simpler analyses this can of course all be doen within one file, but it is good to start practizing. Once you have your own custom-written functions, separate them from the main workflow and load them as needed using `source`.

Formatting and importing data Importing datafiles into R can be a real hazzle in the start. In RStudio there is an 'Import Dataset' button, but this often onvolve excessive clicking and somehow defeauts the purpose of a code-based environments. It's better to figure out a file format that works, and then we have the code and can import the data in a second each time we open the R file.

Good formats for importing into R are .csv and .txt. I generally enter the data in Excel, and then save the file in either of those formats. A couple of points about organizing data in Excel:

1. Avoid empty cells, put `NA` instead. There are ways to make `R` insert `NA`'s in empty cells, but it's always safe to add them already during data entry.
2. Avoid spaces, use underscore (_) instead. For the same reason, this will avoid problems where `R`understands 'body size' as three variables instead of one (body_size). Some also like to keep the units in the variable names, i.e. 'body_size_kg'.

`R` has functions to import many kinds of data. I typically use `read.table` for text files, and `read.csv` for .csv files. On Windows, I often need to use `read.csv2` instead, for some reason. Paya attention to the arguments of these functions, especially things like `sep`, `dec`, and `header` that tells how the data specify string separation, decimal places, and whether there are variable names (column headers) in the data file.

Reproducibility and version control All of this is directly realted to the issue of reproducibility. More and more journals now require code archiving, and by working in a tidy way we are ready to submit our code at the time of submitting the paper.

To take this one step further, we can use a system for version control like GitHub. This also provides us with backup in case our computer dies, and an easy way to publish the code (and data, although this is apparently not the recommended way of publishing data).

For this course, for exmaple, I have made a public GitHub repository avaialble at github.com/oysteiop/QuantitativeAnalysis.

Simple programming

As soon as analyses become a little more complex, some programming skills are very helpful for working in R. First, R in function-based, and we can write our own functions that we can execute over a number of cases. Functions can for example produce a specific plot, fit a specific kind of model, or format a dataframe. It is good practice to use functions whenever we need to repeat a specific procedure multiple times. So the earlier you start practicing, the better.

The perhaps most common programming operation during data analyses is *for*-loops. A *for*-loop performs an operation *for* each element of a index vector.

```r
x = seq(1, 50, 2)

out=NULL
for(i in 1:length(x)){
  out[i] = x[i]*2
}
out
```

```
##  [1]  2  6 10 14 18 22 26 30 34 38 42 46 50 54 58 62 66 70 74 78 82 86 90 94 98
```

Sometimes we want to perform an operation only under certain conditions. We can then use an *if* argument.

```r
out=NULL
for(i in 1:length(x)){
  if(i>10){
    out[i] = x[i]*2
  }
  else{
    out[i] = 0
  }
}
out
```

```
##  [1]  0  0  0  0  0  0  0  0  0  0 42 46 50 54 58 62 66 70 74 78 82 86 90 94 98
```

A shortcut for *if* and *else* is the function `ifelse`.

```r
ifelse(x>25, 1, 0)
```

```
##  [1] 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1
```

A final useful programming argument is *while*, which makes a process continue until a specific condition is reached. For example, this program draws values of 1-10 until the value 21 is reached.

```r
out=0
  while(out<21){
    out = out + floor(runif(1, 1, 10))
    print(out)
  }
```

```
## [1] 8
## [1] 9
## [1] 16
## [1] 17
## [1] 20
## [1] 25
```

Writing functions

R-functions takes arguments, performs operations and, often, returns some results. Base R does not come with a built-in function to compute standard errors, so let's make out own.

```r
computeSE = function(data){
  n = sum(!is.na(data), na.rm=T)
  SE = sqrt(var(data, na.rm=T)/n)
  return(SE)
}
```

Now let's feed some data to our function.

```r
set.seed(1)
x = rnorm(200, 10, 2)
computeSE(x)
```

```
## [1] 0.1313942
```

Some functions can get very complex, with many arguments and running over hundreds of lines of code. This can seem overwhelming, but it is important to keep in mind that functions are generally not written in one go, it is generally best to start from 'inside' the function, with a single line that performs a single operation, and make sure it works before adding complexity.