Processing and Analysis of Biological Data

The Linear Model 1: Linear regression

Øystein H. Opedal

31 Oct 2023

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

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

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. 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.

In the code below, note that `R` functions know the order of arguments, and that second incidence of `rnorm` below skips the formalities. To know the names and order of arguments to functions, call `?function`.

```
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="Leaf length (mm)",
     ylab="Leaf width (mm)")
```

At this point let's talk about how to write `R` code. In the `R` chunk 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 pleasant 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 appearance of `R`-plots, see the wonderful world of `par()`. If you prefer, you can choose to learn alternative plotting frameworks such as `ggplot`, but in these lecture notes I 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

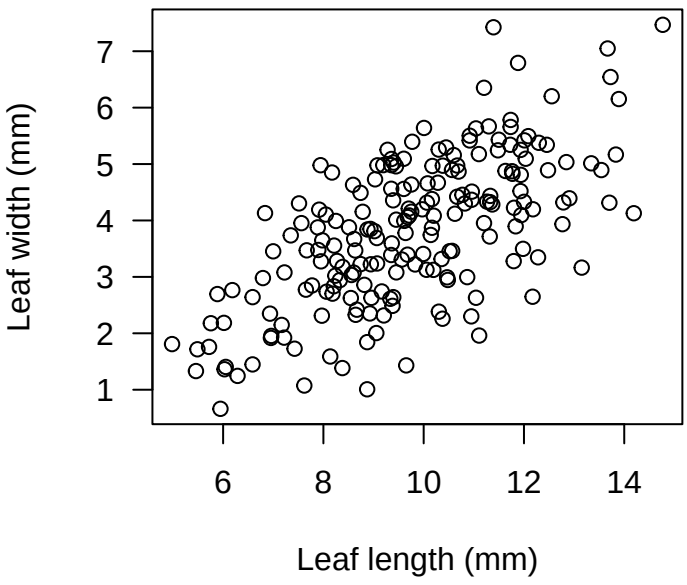


Figure 1: Scatterplot of simulated data from a linear model

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)
```

The object `m` now holds the fitted model. Let's first extract the model coefficients and produce some plots of the residuals. To do so we first pull out the parameter estimates from the fitted model. In general, to extract a specific value or component of an object, we can call `str()`, which will return a list of components available within the object. We then use `$` to index the component we want to extract (here `m$coef`).

```
cf = m$coef
cf
```

```
## (Intercept)           x
##  -0.4011431   0.4333027
```

EXERCISE: Use the parameter estimates to draw (by hand) the scatterplot of y vs x with the regression line, indicating the location of the intercept and roughly the correct slope.

To obtain the predicted values, we pull out (using square brackets `[]` to index elements of a vector) the relevant parameter estimates (intercept and slope), and compute the predicted values using the equation for the linear model we fitted. Below we also add lines from the predicted values to the actual value of the response variable, illustrating the residuals.

```
predvals = cf[1] + cf[2]*x

par(mfrow=c(1,2))
plot(x, y, las=1,
     xlab="Leaf length (mm)",
     ylab="Leaf width (mm)")
abline(m)
segments(x, y, x, predvals)
hist(residuals(m), xlab="", las=1)
```

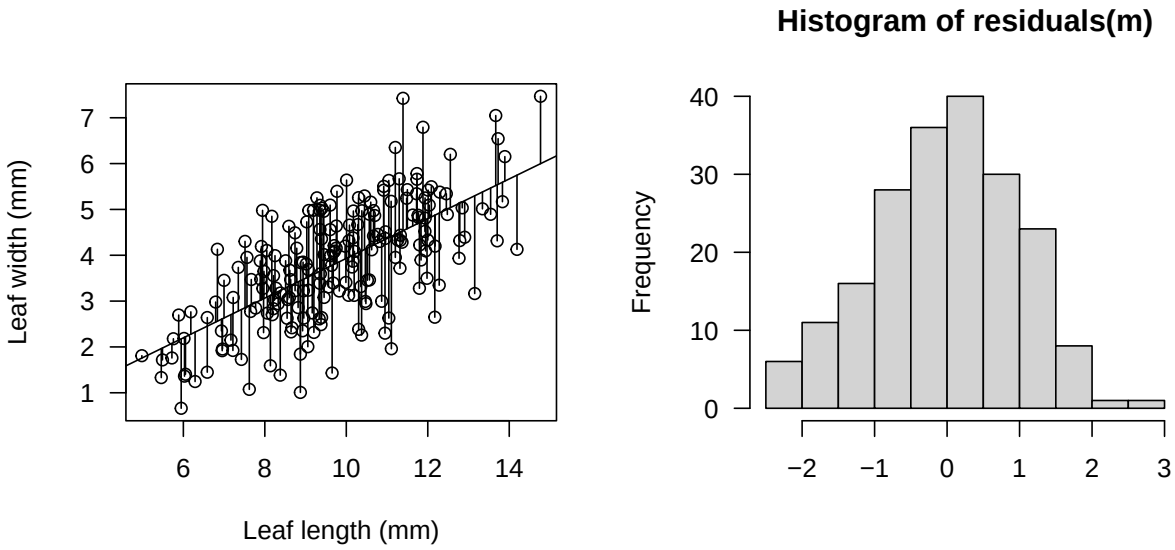


Figure 2: Scatterplot with fitted regression line and illustration of residuals

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))`. Notice 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.

Above we also added the fitted regression line through the ready-made `abline` function. To keep the regression line within the data range, it is better to make predictions for new values of x spanning the data range and add the regression line using the `lines` function.

```
newx = seq(min(x), max(x), length.out=200)
predy = cf[1] + cf[2]*newx

plot(x, y, las=1,
     xlab="Leaf length (mm)",
     ylab="Leaf width (mm)")
lines(newx, predy)
```

After checking that the residuals are fine, let's now have a look at the results of our linear model fit. The `summary` function is also generic.

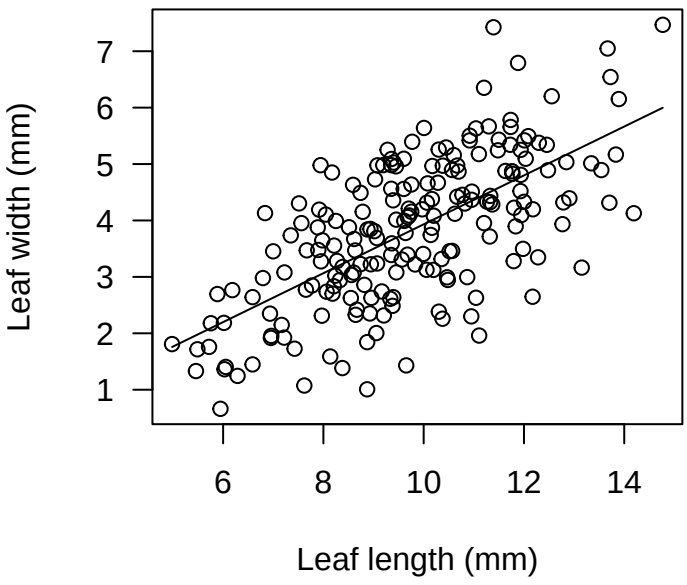


Figure 3: Scatterplot with fitted regression line

```
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, produce a data frame holding the `x` and `y` values, sample from this dataset (with replacement), fit the model, and save each estimate for the slope of y on x. The samples will give the sampling distribution for the slope, and its standard deviation will provide an estimate of the standard error.

```
df = data.frame(x, y)
head(df)
```

```
##           x        y
## 1  9.971266 4.197632
## 2  8.872600 1.007507
## 3 10.375599 2.258196
## 4  8.136517 1.588956
## 5  7.898157 3.476802
## 6  5.885831 2.695560
```

```
## [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. Recall that the regression slope is given by the ratio of the covariance between y and x, and the variance in x, $Cov(y, x)/Var(x)$.

```
cov(y,x)/var(x)
```

```
## [1] 0.4333027
```

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. To do so we compute the difference in the predicted values when x is at its mean (second term below), and when x in at its mean + one standard deviation (first term below). Why can we ignore the intercept here?

```
(cf[2]*(mean(x) + sd(x))) - (cf[2]*mean(x))
```

```
##         x
## 0.8606095
```

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

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 properties 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 $\hat{y}$,

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

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

```
y_hat = cf[1] + cf[2]*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.

```
cf[2]^2*var(x)
```

```
##         x
## 0.7406487
```

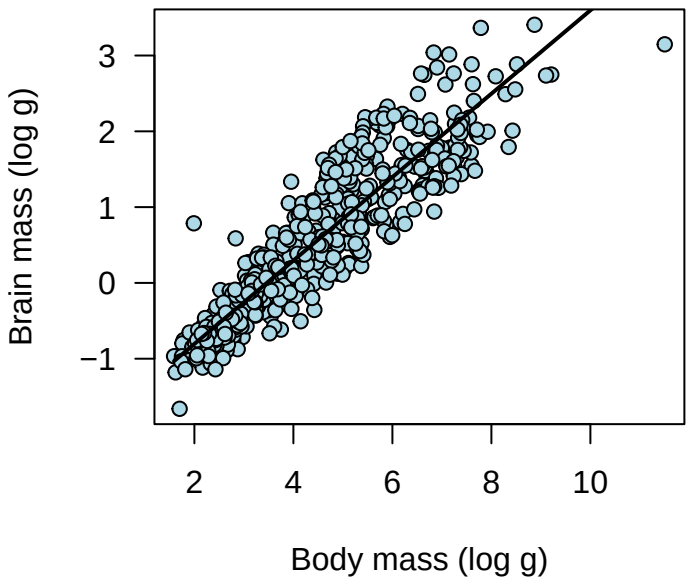
Exercise: fitting a linear regression to real data

Choose any dataset you may have involving a continuous response variable and a continuous predictor. Fit a simple linear regression, interpret the results, produce a nice figure including the fitted regression line, and write simple methods and results presenting the analysis and results.

If you don't have any data, use the dataset `bird_allometry` in the datasets folder. This dataset contains body mass and brain mass for males and females of different bird species. The scaling of brain size (or other body parts) with body size is referred to as the study of allometry, and you may want to read about these analyses before fitting your models. As a hint, the scaling of parts of a body with body size is expected to follow a power-law relationship on the form $y = ax^b$, which can be linearized through the logarithmic transformation $log(y) = log(a) + b \times log(x)$.

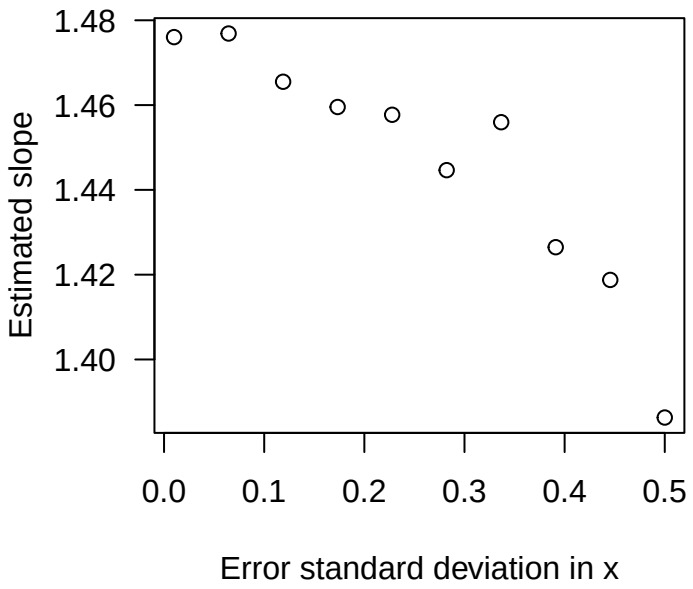
```
birds = read.csv("datasets/allometry/bird_allometry.csv")
head(birds)
```

```
##                Genus_Species Sex brain_mass body_mass
## 1         Accipiter_gentilis   f   7.686143 1049.1571
## 2         Accipiter_gentilis   m   7.618500  678.2833
## 3            Accipiter_nisus   f   3.112797  252.1263
## 4            Accipiter_nisus   m   2.637390  136.1441
## 5         Accipiter_striatus   f   5.700000  520.0000
## 6 Acridotheres_cristatellus   m   2.310000  122.3800
```



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

The standard linear model assumes that the predictor variable is measured without 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. As always with programming exercises, start by performing the necessary operations once, before building loops or functions. Here, you can start by simulating some data, and fit the model with no measurement error. Then, add some error, and see what happens to the slope estimate.



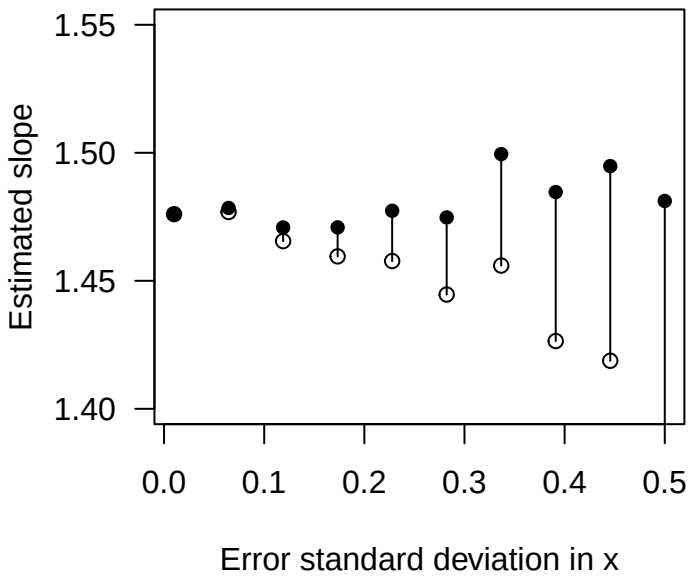
For a simple model like this, the expected attenuation bias (downward bias) in the slope can be estimated by the reliability ratio

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

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.



What about error in the response variable? Repeat the exercise with error in y instead of error in x. What happens?