Introduction to Regression

MAP 533 - Regression,Fall 2018

Septembre 2018

Introduction

Regression analysis is about investigating quantitative, predictive relationships between variables. Its about situations where there is some sort of link, tie or relation between two (or more) variables, so if we know the value of one of them, it tells us something about the other. The concrete sign of this is that knowledge of one variable lets us predict the target variable better than if we didnt know the other. The main goal of this class is to learn how to build predictive mathematical models, seeing whether such models really have any predictive power, and comparing their predictions.

This text is largely inspired by the lecture notes on regression by A. Guyader and S. Cosma as well as the monograph by J.F. Monaham, *A Primer on Linear Models.*

Contents

Chapter 1

The simple linear model

1.1 Motivation

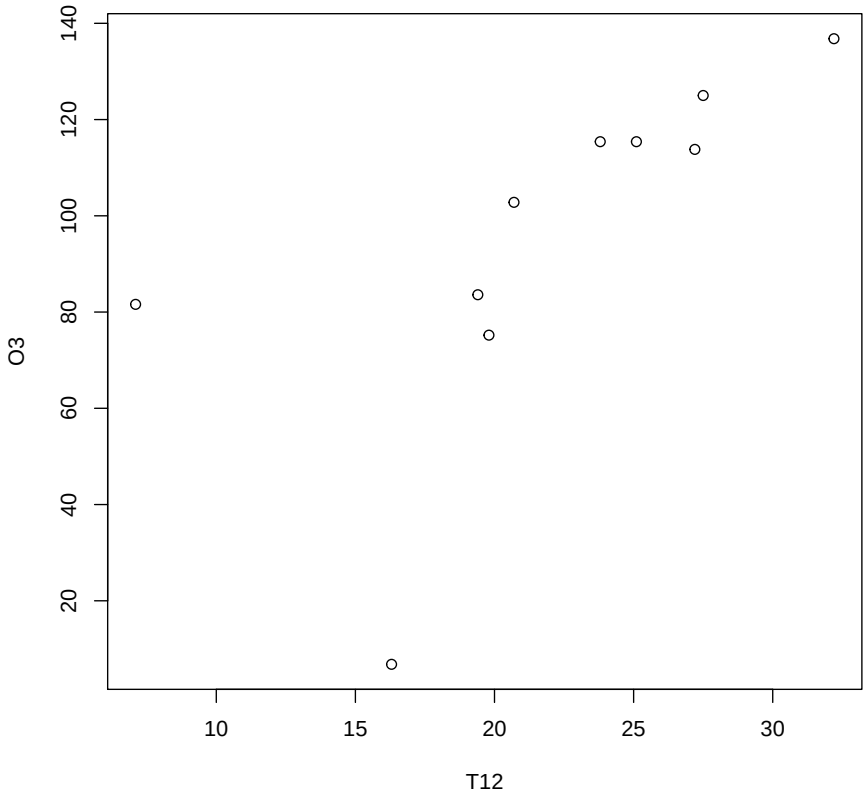
Let's start with an example. For the sake of public health, we monitor the concentration of O_3 in the atmosphere. We are interested in determining whether the maximum daily concentration in O_3 can be explained by the daily temperature at noon. We have access to the following data: $n = 10$ daily observations of temperature at noon and maximum 0_3 concentration

Temperature	23.8	16.3	27.2	7.1	25.1	27.5	19.4	19.8	32.2	20.7
O_3 max	115.4	76.8	113.8	81.6	115.4	125	83.6	75.2	136.8	102.8

```
T12 <- c(23.8, 16.3, 27.2, 7.1, 25.1, 27.5, 19.4, 19.8, 32.2, 20.7)
O3 <- c(115.4, 6.8, 113.8, 81.6, 115.4, 125, 83.6, 75.2, 136.8, 102.8)
```

It is always a good idea to visualize the data first

```
plot(O3 ~ T12)
```



This plot seems to show some relation between x_i (temperature) and y_i (O_3) of the form

$$y_i \approx f(x_i), \quad i = 1, \dots, 10.$$

Here the function f is called regression function. It gives us a prediction for y_i based on observation x_i. We can take more or less complex functions f to build our model. Therefore, we need a criterion to compare predictions.

We consider the Mean Squared Error (MSE) of f

$$MSE(f) = \mathbb{E}[(Y - f(X))^2].$$

Let looks at what happens when we use a constant function $f(x) = m$ for some fixed $m \in \mathbb{R}$. In other word, we do not take into account the value of covariate X. We can decompose build the **bias-variance decomposition** of the MSE:

$$MSE(f) = \mathbb{E}[(\mathbb{E}[Y] - m)^2] + \mathrm{Var}(Y).$$

It is obvious to see that the MSE is minimized for $m = \mathbb{E}[Y]$. If we consider now an arbitrary function $f(x)$, we can use conditional expectations to reduce this minimization problem to the case of constant functions.

$$\mathbb{E}[(Y - f(X))^2] = \mathbb{E}[\mathbb{E}[(Y - f(X))^2|X]]. \tag{1.1}$$

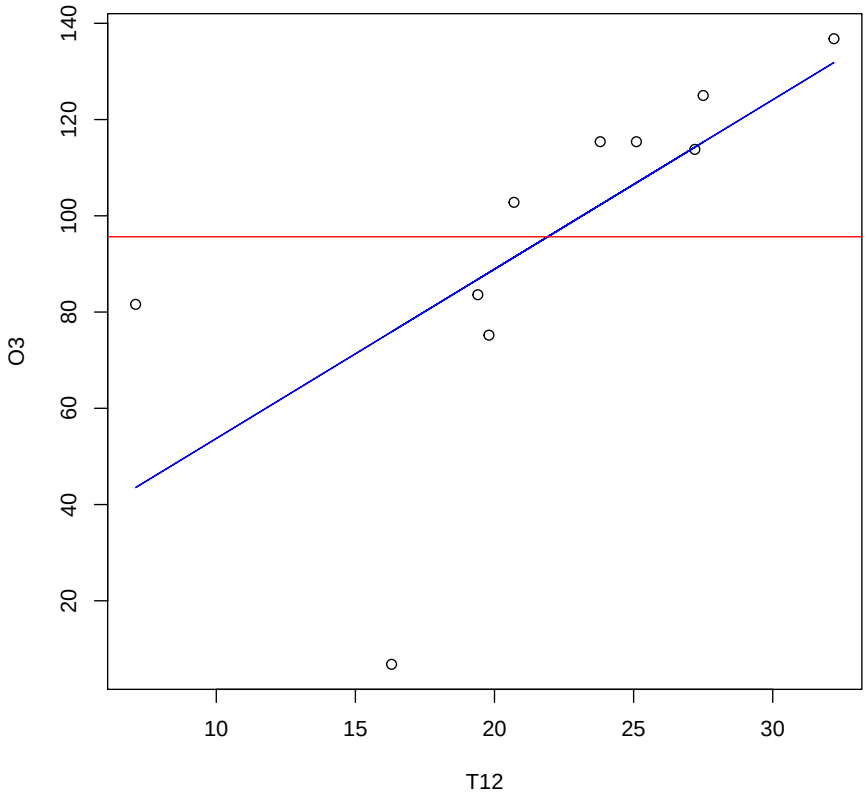
Applying the same bias-variance decompostion, we see that the function minimizing the above display is

$$\mu(x) = \mathbb{E}[Y|X = x]. \tag{1.2}$$

It is called the regression function of Y on X. Ideally, this is the function we want to recover based on our data. Sometimes this function is really complicated and we build simpler approximation of it. For instance $\mu(\cdot)$ is a constant function, affine function, polynomial of degree d. This approximation may not be true but the important aspect is to understand what is the best approximation (prediction) of our data within a class of models.

We can draw the best constant approximation and the best affine approximation

```
flin <- lm(O3 ~ T12)
flinpred <- predict.lm(flin)
plot(T12, O3)
lines(T12, flinpred, type = "l", col = "blue")
abline(a = mean(O3), b = 0, col = "red")
```



Visually, it appears that the affine approximation provides a better fit to the data. This course will consist in properly formalizing this observation.

We consider the family of linear models $f(x) = b_0 + b_1 x$. The MSE of these affine functions is given by

$$\begin{aligned} MSE(f) = MSE(b_0, b_1) &= \mathbb{E}[(Y - b_0 - b_1 X)^2] \\ &= \mathbb{E}[Y^2] - 2b_0\mathbb{E}[Y] - 2b_1 \mathrm{Cov}(X, Y) - 2b_1\mathbb{E}[X]\mathbb{E}[Y] \\ &\qquad + b_0^2 + 2b_0 b_1 \mathbb{E}[X] + b_1^2 \mathrm{Var}(X) + b_1^2 (\mathbb{E}[X])^2. \end{aligned}$$

This is a function of parameters b_0, b_1. Let observe how the MSE varies as a function of b_0, b_1 on an example:

It is possible to find a close form solution for the problem

$$\min_{b_0, b_1} MSE(b_0, b_1).$$

In other words, we can have access in theory to the best linear approximation.

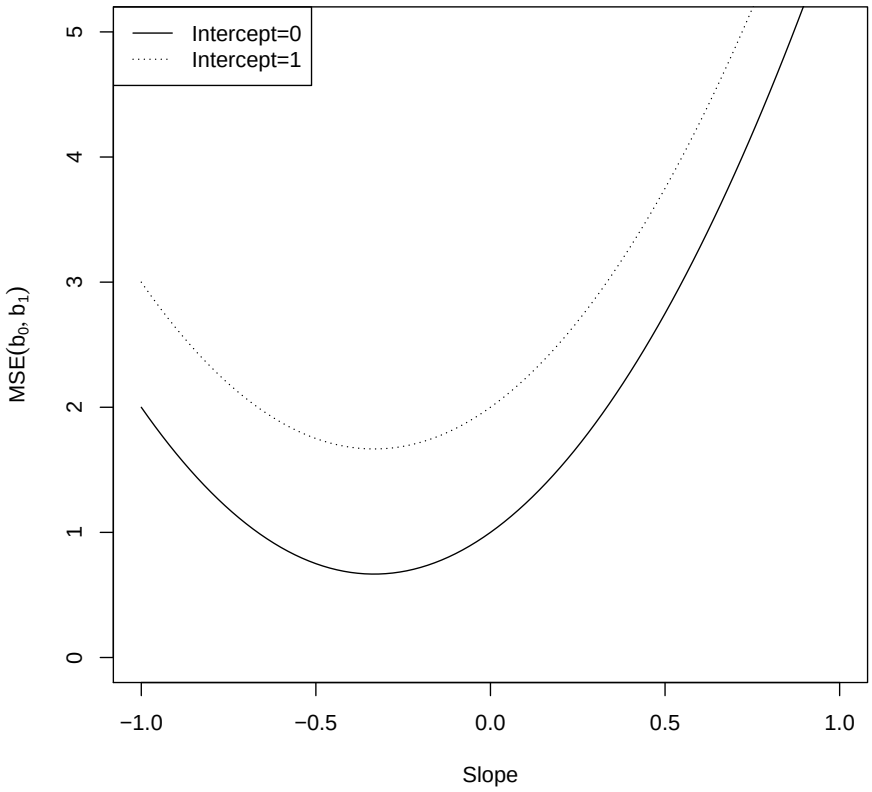


FIGURE 1.1: *Mean squared error of linear models with different slopes and intercepts, when* $\mathbb{E}[X] = \mathbb{E}[Y] = 0$, $\mathrm{Var}(X) = 3, \mathrm{Var}(Y) = 1, \mathrm{Cov}(X, Y) = 1$. *Each curve represents a different intercept* b_0 *in the linear model* $b_0 + b_1 X$ *for* Y

It is given by

$$b_1^* = \frac{\mathrm{Cov}(X,Y)}{\mathrm{Var}(X)}, \quad \text{and} \quad b_0^* = \mathbb{E}[Y] - b_1^* \mathbb{E}[X].$$

Comments:

- Note that computing the solution (b_0^*, b_1^*) requires some knowledge on the joint distribution of (X, Y) which is unrealistic in many practical situations. We will need to use a sample to estimate these quantities.
- We never assumed that the true regression function $\mu(x)$ is linear. We simply studied in the above example the approximation of μ by a linear function. In some cases, this approximation will be quite precise, in some other cases it will be a poor approximation. For instance, the periodic function $\mu(x) = \sin(x)$ is poorly approximated by a linear function.

The simple linear regression model is a basic statistical model that is used to predict a random variable Y from another one X. Let $(X, Y) \in \mathbb{R}^2$ be a pair of random variables satisfying

$$\mathbb{E}[Y|X] = \beta_0 + \beta_1 X. \tag{1.3}$$

We assume that the noise $\epsilon = Y - \mathbb{E}[Y|X]$ is a zero mean random variable with finite variance σ^2 and is independent of X. The noise variable may represent measurement error, or fluctuations in Y, or some combination of both.

Note that these assumptions are non-trivial and need to be checked in applications:

- The assumption of additive noise; We could indeed imagine multiplicative measurement error.
- The assumption of a linear form for the relation between Y and X; We could have a non-linear relation in many applications.
- The noise *homoskedasticity* assumption, i.e. constant variance.

Nevertheless, this model is relevant and useful in many applications.

1.2 Plug-In Estimates

Recall that the optimal linear predictor of Y from X has slope $\beta_1 = \mathrm{Cov}[X, Y] / \mathrm{Var}[X]$. Thus β_1 is not directly accessible when the distribution is unknown. We observe n independent data points, say $(x_1, y_1), (x_2, y_2), \ldots (x_n, y_n)$. How shall we *estimate* β_1 from this data set?

A natural approach would be to use the data to build the *sample* covariance and *sample* variance, and then take their ratio. The sample variance of X is

$$s_X^2 = \frac{1}{n} \sum_{i=1}^{n} (x_i - \overline{x})^2,$$

and the sample covariance is

$$c_{XY} = \frac{1}{n}\sum_{i=1}^{n}(x_i - \overline{x})(y_i - \overline{y}),$$

where $\overline{x} = \sum_{i=1}^{n} x_i/n$, and similarly for other variables. Then, we get the following **sample** slope

$$\widehat{\beta_1} = \frac{c_{XY}}{s_X^2}$$

This is what we call the **plug-in principle** where we find equations for the parameters which would hold if we knew the full distribution, and "plug in" the sample versions of the population quantities. Under some conditions, we also proved the consistency of the plug-in estimators of β_1n and β_0.

We hope at the very least that the estimator $\widehat{\beta_1}$ is **consistent**. That is that it converges to β_1 as $n \to \infty$. This raises the natural question of necessary and sufficient conditions to get $\widehat{\beta_1} \to \beta_1$?

Lemma 1. *Assume that* $c_{XY} \to \mathrm{Cov}(X,Y)$ *and* $s_X^2 \to \mathrm{Var}(X)$ *as* $n \to \infty$. *Then* $\widehat{\beta_1}$ *is consistent.*

Proof. We have

$$c_{XY} = \frac{1}{n}\sum_{i=1}^{n} x_i y_i - \bar{x}\bar{y} \tag{1.4}$$

According to the model, $y_i = (\beta_0 + \beta_1 x_i + \epsilon_i)$. So (after a little more algebra)

$$\begin{aligned} c_{XY} &= \frac{1}{n}\sum_{i=1}^{n} x_i(\beta_0 + \beta_1 x_i + \epsilon_i) - \overline{x}\overline{\beta_0 + \beta_1 x + \epsilon} && (1.5)\\ &= \beta_0\overline{x} + \beta_1\overline{x^2} + \overline{x\epsilon} - \overline{x}\beta_0 - \beta_1\overline{x}^2 - \bar{x}\bar{\epsilon} && (1.6)\\ &= \beta_1 s_X^2 + \overline{x\epsilon} - \bar{x}\bar{\epsilon} && (1.7) \end{aligned}$$

Because ϵ has mean 0, as $n \to \infty$, the law of large numbers says $\bar{\epsilon} \to 0$. Because ϵ is uncorrelated with x, using the law of large numbers again says that $\overline{x\epsilon} \to 0$ as well. So

$$\widehat{\beta_1} = \frac{c_{XY}}{s_X^2} \to \beta_1.$$

□

Consistency is the weakest property that we want for our estimator. We especially would like to quantify the rate of convergence. That is for a given sample size, how far from the truth our estimate is likely to be. In what follows, we will develop more precise statistical inference results for the prediction of Y, the estimation of β and testing hypothesis.

1.3 Least Squares Estimates

An alternative way of estimating the simple linear regression model starts from the objective we are trying to reach, rather than from the formula for the slope. The true optimal slope and intercept are the ones which minimize the mean squared error:

$$(\beta_0, \beta_1) = \text{argmin}_{(b_0,b_1)} \mathbb{E}\left[(Y - (b_0 + b_1 X))^2\right] \tag{1.8}$$

This is a function of the complete distribution, so we can't get it from data, but we can approximate it with data. The **in-sample**, **empirical** or **training** MSE is

$$\widehat{MSE}(b_0, b_1) \equiv \frac{1}{n}\sum_{i=1}^{n}(y_i - (b_0 + b_1 x_i))^2 \tag{1.9}$$

Notice that this is a function of b_0 and b_1; it is also, of course, a function of the data, $(x_1, y_1), (x_2, y_2), \ldots (x_n, y_n)$, but we will generally suppress that in our notation.

If our samples are all independent, for any fixed (b_0, b_1), the law of large numbers tells us that $\widehat{MSE}(b_0, b_1) \to MSE(b_0, b_1)$ as $n \to \infty$. So it doesn't seem unreasonable to try minimizing the in-sample error, which we can compute, as a proxy for minimizing the true MSE that is not directly accessible.

Start by taking the derivatives with respect to the slope and the intercept:

$$\frac{\partial \widehat{MSE}}{\partial b_0} = -\frac{2}{n}\sum_{i=1}^{n}(y_i - (b_0 + b_1 x_i)) \tag{1.10}$$

$$\frac{\partial \widehat{MSE}}{\partial b_0} = -2\frac{2}{n}\sum_{i=1}^{n}(y_i - (b_0 + b_1 x_i))x_i \tag{1.11}$$

Set these to zero at the optimum $(\hat{\beta}_0, \hat{\beta}_1)$:

$$\begin{aligned} \frac{1}{n}\sum_{i=1}^{n}(y_i - (\hat{\beta}_0 + \hat{\beta}_1 x_i)) &= 0 \\ \frac{1}{n}\sum_{i=1}^{n}(y_i - (\hat{\beta}_0 + \hat{\beta}_1 x_i))x_i &= 0 \end{aligned} \tag{1.12}$$

These are often called the **normal equations** for least-squares estimation, or the **estimating equations**: a system of two equations in two unknowns, whose solution gives the estimate. Equivalently, we obtain:

$$\hat{\beta}_0 + \hat{\beta}_1 \overline{x} = \overline{y} \tag{1.13}$$

$$\hat{\beta}_0 \overline{x} + \hat{\beta}_1 \overline{x^2} = \overline{xy} \tag{1.14}$$

Solving this system of linear equations, we get

$$\hat{\beta}_0 = \overline{y} - \hat{\beta}_1 \overline{x} \tag{1.15}$$

Substituting this into the remaining equation,

$$\hat{\beta}_1 = \frac{c_{XY}}{s_X^2}. \tag{1.16}$$

That is, the least-squares estimate of the slope is our old friend the plug-in estimate of the slope, and thus the least-squares intercept is also the plug-in intercept.

Remark 1. *In general, the plug-in estimator and the least-squares estimator are distinct for other models.*

1.4 Bias, Variance and Standard Error of Parameter Estimates

Whether we think of it as deriving from pluging-in or from least squares, we work out some of the properties of this estimator of the coefficients, using the model assumptions. We'll start with the slope, $\hat{\beta}_1$.

$$\begin{aligned}
\hat{\beta}_1 &= \frac{c_{XY}}{s_X^2} && (1.17)\\
&= \frac{\frac{1}{n}\sum_{i=1}^n x_i y_i - \bar{x}\bar{y}}{s_X^2} && (1.18)\\
&= \frac{\frac{1}{n}\sum_{i=1}^n x_i(\beta_0 + \beta_1 x_i + \epsilon_1) - \bar{x}(\beta_0 + \beta_1\bar{x} + \bar{\epsilon})}{s_X^2} && (1.19)\\
&= \frac{\beta_0\bar{x} + \beta_1\overline{x^2} + \frac{1}{n}\sum_{i=1}^n x_i\epsilon_i - \bar{x}\beta_0 - \beta_1\bar{x}^2 - \bar{x}\bar{\epsilon}}{s_x^2} && (1.20)\\
&= \frac{\beta_1 s_X^2 + \frac{1}{n}\sum_{i=1}^n x_i\epsilon_i - \bar{x}\bar{\epsilon}}{s_X^2} && (1.21)\\
&= \beta_1 + \frac{\frac{1}{n}\sum_{i=1}^n x_i\epsilon_i - \bar{x}\bar{\epsilon}}{s_X^2} && (1.22)
\end{aligned}$$

Since $\bar{x}\bar{\epsilon} = n^{-1}\sum_i \bar{x}\epsilon_i$,

$$\hat{\beta}_1 = \beta_1 + \frac{\frac{1}{n}\sum_{i=1}^n (x_i - \bar{x})\epsilon_i}{s_X^2} \tag{1.23}$$

This representation of the slope estimate shows that it is equal to the true slope (β_1) plus something which depends on the noise terms (the ϵ_i, and their sample average $\bar{\epsilon}$). We'll use this to find the expected value and the variance of the estimator $\hat{\beta}_1$.

In the next couple of paragraphs, we will be assumed that the x_i are fixed deterministic constants. This is appropriate in designed experiments, where we get to chose their value. In randomized experiments or in observational

studies, the x_i aren't necessarily fixed; however, these expressions will be correct for the conditional expectation $\mathbb{E}\left[\hat{\beta}_1 | x_1, \ldots x_n\right]$ and conditional variance $\mathrm{Var}\left[\hat{\beta}_1 | x_1, \ldots x_n\right]$. We will see how to get the unconditional expectation and variance from these expressions.

Expected value and bias The **bias** of an estimator is the difference between its expected value and the truth. If this difference is zero, then we say that this estimator is unbiased.

Recall that $\mathbb{E}\left[\epsilon_i | X_i\right] = 0$, so

$$\frac{1}{n}\sum_{i=1}^{n}(x_i - \bar{x})\mathbb{E}\left[\epsilon_i\right] = 0 \tag{1.24}$$

Hence, the bias of $\hat{\beta}_1$ satisfies

$$\mathrm{bias}(\hat{\beta}_1) := \mathbb{E}[\hat{\beta}_1] - \beta_1 = \frac{1}{n}\sum_{i=1}^{n}(x_i - \bar{x})\mathbb{E}\left[\epsilon_i\right] = 0. \tag{1.25}$$

We conclude that $\hat{\beta}_1$ is an **unbiased** estimator of the optimal slope.

Remark 2. *Note that unbiased estimators are not necessarily superior to biased ones. The total error depends on both the bias of the estimator and its variance (recall the bias-variance decomposition of Chapter 1). There are many situations where you can remove lots of bias at the cost of adding a little variance.*

Turning to the intercept,

$$\begin{aligned}
\mathbb{E}\left[\hat{\beta}_0\right] &= \mathbb{E}\left[\overline{Y} - \hat{\beta}_1\overline{X}\right] && (1.26)\\
&= \beta_0 + \beta_1\overline{X} - \mathbb{E}\left[\hat{\beta}_1\right]\overline{X} && (1.27)\\
&= \beta_0 + \beta_1\overline{X} - \beta_1\overline{X} && (1.28)\\
&= \beta_0 && (1.29)
\end{aligned}$$

so it, too, is unbiased.

Variance and Standard Error Using the formula for the variance of a sum, and the model assumption that all the ϵ_i are uncorrelated with each other,

$$\begin{aligned}
\mathrm{Var}\left[\hat{\beta}_1\right] &= \mathrm{Var}\left[\beta_1 + \frac{\frac{1}{n}\sum_{i=1}^{n}(x_i - \bar{x})\epsilon_i}{s_X^2}\right] && (1.30)\\
&= \mathrm{Var}\left[\frac{\frac{1}{n}\sum_{i=1}^{n}(x_i - \bar{x})\epsilon_i}{s_X^2}\right] && (1.31)\\
&= \frac{\frac{1}{n^2}\sum_{i=1}^{n}(x_i - \bar{x})^2 \mathrm{Var}\left[\epsilon_i\right]}{(s_X^2)^2} && (1.32)\\
&= \frac{\frac{\sigma^2}{n} s_X^2}{(s_X^2)^2} && (1.33)\\
&= \frac{\sigma^2}{n s_X^2} && (1.34)
\end{aligned}$$

In words, this says that the variance of the slope estimate goes up as the noise around the regression line (σ^2) gets bigger, and goes down as we have more observations (n), which are further spread out along the horizontal axis (s_X^2); it should not be surprising that it's easier to work out the slope of a line from many, well-separated points on the line than from a few points smushed together.

The **standard error** of an estimator is just its standard deviation, or the square root of its variance:

$$\mathrm{se}(\hat{\beta}_1) = \frac{\sigma}{\sqrt{n s_X^2}} \quad (1.35)$$

Exercise 1. *Derive the standard deviation of $\hat{\beta}_0$.*

Unconditional-on-X Properties In the previous paragraphs, we have looked at the expectation and variance of $\hat{\beta}_1$ conditional on $x_1, \ldots x_n$, either because the x's really are non-random (e.g., controlled by us), or because we're just interested in conditional inference. It is a rather easy task to deduce from those results unconditional properties in the simple linear regression model. Indeed, to get the unconditional expectation, we use the law of total expectation:

$$\begin{aligned}
\mathbb{E}\left[\hat{\beta}_1\right] &= \mathbb{E}\left[\mathbb{E}\left[\hat{\beta}_1 | X_1, \ldots X_n\right]\right] && (1.36)\\
&= \mathbb{E}\left[\beta_1\right] = \beta_1 && (1.37)
\end{aligned}$$

That is, the estimator is *unconditionally* unbiased.

Similarly, to get the unconditional variance, we use the law of total variance:

$$\begin{aligned}
\mathrm{Var}\left[\hat{\beta}_1\right] &= \mathbb{E}\left[\mathrm{Var}\left[\hat{\beta}_1 | X_1, \ldots X_n\right]\right] + \mathrm{Var}\left[\mathbb{E}\left[\hat{\beta}_1 | X_1, \ldots X_n\right]\right] && (1.38)\\
&= \mathbb{E}\left[\frac{\sigma^2}{n s_X^2}\right] + \mathrm{Var}\left[\beta_1\right] && (1.39)\\
&= \frac{\sigma^2}{n}\mathbb{E}\left[\frac{1}{s_X^2}\right] && (1.40)
\end{aligned}$$

1.5 Parameter Interpretation

Two of the parameters are easy to interpret.

σ^2 is the variance of the noise around the regression line; σ is a typical distance of a point from the line. ("Typical" here in a special sense, it's the root-mean-squared distance, rather than, say, the average absolute distance.)

β_0 is the simply the expected value of Y when X is 0, $\mathbb{E}[Y|X=0]$. The point $X=0$ usually has no special significance, but this setting does ensure that the line goes through the point $(\mathbb{E}[X], \mathbb{E}[Y])$.

The interpretation of the slope is also straightforward. We have for any x

$$\beta_1 = \mathbb{E}[Y|X=x] - \mathbb{E}[Y|X=x-1] \tag{1.41}$$

or, for any x_1, x_2,

$$\beta_1 = \frac{\mathbb{E}[Y|X=x_2] - \mathbb{E}[Y|X=x_1]}{x_2 - x_1} \tag{1.42}$$

We can interpret β_1 as follows. If we select two sets of cases from the unmanipulated distribution where X differs by 1, then on average Y will change by β_1.

1.6 Prediction in the Simple Linear Regression model

In the prediction problem, we are not so much interested in the values of β_0 and β_1 themselves, but rather want to use those values to predict Y from X. How do we make those predictions, and how good are they?

If we knew β_0 and β_1, and that $X=x$, then our prediction for Y would be $\beta_0 + \beta_1 x$. This is, assuming the simple linear regression model is true, that is

$$\mathbb{E}[Y|X=x] = \beta_0 + \beta_1 x,$$

which we recall is the best prediction we can make assuming we know enough of the joint distribution of (X,Y).

Since we do not know β_0 and β_1, we use our estimates of the coefficients. At an *arbitrary* value of X, say x, we predict that on average Y will be

$$\hat{m}(x) = \hat{\beta}_0 + \hat{\beta}_1 x \tag{1.43}$$

This point prediction is called the **fitted value** at x.

Notice the fitted value $\hat{m}(x)$ is an *estimate* of $\mathbb{E}[Y|X=x] = \beta_0 + \beta_1 x$. But $\hat{m}(x)$ is a function of our data, which are random, hence $\hat{m}(x)$ is also random. It inherits its randomness from $\hat{\beta}_0$ and $\hat{\beta}_1$.

To analyze the randomness in $\hat{m}(x)$, we will represent it as constants plus a weighted sum of uncorrelated noise terms. Using Eqs. **??**,

$$\begin{aligned}
\hat{m}(x) &= \hat{\beta}_0 + \hat{\beta}_1 x && (1.44)\\
&= \overline{y} - \hat{\beta}_1 \overline{x} + \hat{\beta}_1 x && (1.45)\\
&= \overline{y} + (x - \overline{x})\hat{\beta}_1 && (1.46)
\end{aligned}$$

Combining the previous display with Eq. 1.23 we get sample mean,

$$\begin{aligned}
\hat{m}(x) &= \frac{1}{n}\sum_{i=1}^{n} y_i + (x - \overline{x})\left(\beta_1 + \sum_{i=1}^{n} \frac{x_i - \overline{x}}{n s_X^2}\epsilon_i\right) && (1.47)\\
&= \frac{1}{n}\sum_{i=1}^{n} (\beta_0 + \beta_1 x_i + \epsilon_i) + (x - \overline{x})\left(\beta_1 + \sum_{i=1}^{n} \frac{x_i - \overline{x}}{n s_X^2}\epsilon_i\right) && (1.48)\\
&= \beta_0 + \beta_1 \overline{x} + \frac{1}{n}\sum_{i=1}^{n} \epsilon_i + (x - \overline{x})\beta_1 + (x - \overline{x})\sum_{i=1}^{n} \frac{x_i - \overline{x}}{n s_X^2}\epsilon_i && (1.49)\\
&= \beta_0 + \beta_1 x + \frac{1}{n}\sum_{i=1}^{n}\left(1 + (x - \overline{x})\frac{x_i - \overline{x}}{s_X^2}\right)\epsilon_i && (1.50)
\end{aligned}$$

Now we can check that our predictions are unbiased:

$$\begin{aligned}
\mathbb{E}\left[\hat{m}(x)\right] &= \mathbb{E}\left[\beta_0 + \beta_1 x + \frac{1}{n}\sum_{i=1}^{n}\left(1 + (x - \overline{x})\frac{x_i - \overline{x}}{s_X^2}\right)\epsilon_i\right] && (1.51)\\
&= \beta_0 + \beta_1 x + \frac{1}{n}\sum_{i=1}^{n}\left(1 + (x - \overline{x})\frac{x_i - \overline{x}}{s_x^2}\right)\mathbb{E}\left[\epsilon_i\right] && (1.52)\\
&= \beta_0 + \beta_1 x && (1.53)
\end{aligned}$$

We compute now the variance of these predictions:

$$\begin{aligned}
\mathrm{Var}\left[\hat{m}(x)\right] &= \mathrm{Var}\left[\beta_0 + \beta_1 x + \frac{1}{n}\sum_{i=1}^{n}\left(1 + (x - \overline{x})\frac{x_i - \overline{x}}{s_X^2}\right)\epsilon_i\right] && (1.54)\\
&= \mathrm{Var}\left[\frac{1}{n}\sum_{i=1}^{n}\left(1 + (x - \overline{x})\frac{x_i - \overline{x}}{s_X^2}\right)\epsilon_i\right] && (1.55)\\
&= \frac{1}{n^2}\sum_{i=1}^{n}\left(1 + (x - \overline{x})\frac{x_i - \overline{x}}{s_X^2}\right)^2 \mathrm{Var}\left[\epsilon_i\right] && (1.56)\\
&= \frac{\sigma^2}{n^2}\sum_{i=1}^{n} 1 + 2(x - \overline{x})\frac{x_i - \overline{x}}{s_X^2} + (x - \overline{x})^2\frac{(x_i - \overline{x})^2}{s_X^4} && (1.57)\\
&= \frac{\sigma^2}{n^2}\left(n + 0 + (x - \overline{x})^2\frac{n s_X^2}{n s_X^4}\right) && (1.58)\\
&= \frac{\sigma^2}{n}\left(1 + \frac{(x - \overline{x})^2}{s_X^2}\right) && (1.59)
\end{aligned}$$

Notice what's going on here:

- The variance grows as σ^2 grows: the more noise there is around the regression line, the harder we find it to estimate the regression line, and the more of that noise will propagate into our predictions.
- The larger n is, the smaller the variance: the more points we see, the more exactly we can pin down the line, and so our predictions.
- The variance of our predictions is the sum of two terms. The first, which doesn't depend on x, is σ^2/n, which is the variance of $\overline{y}$.
- The other term does change with x, specifically with $(x-\overline{x})^2$: the further our operating point x is from the center of the data $\overline{x}$, the bigger our uncertainty. This is the uncertainty that comes with being unable to pin down the slope precisely; it therefore shrinks with s_X^2, since it's easier to find the slope when the points have a wide spread on the horizontal axis.

Again, Eq. 1.59 is conditional on the x_i. If those are random, we need to use the law of total variance to get the unconditional variance of $\hat{m}(x)$.

1.6.1 Estimating the variance σ^2

Under the simple linear regression model, it is easy to show (Exercise!) that

$$\mathbb{E}\left[(Y - (\beta_0 + \beta_1 X))^2\right] = \sigma^2 \tag{1.60}$$

This suggests that the minimal value of the sample MSE,

$$\hat{\sigma}^2 = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{m}(x_i))^2 \tag{1.61}$$

is a natural estimator for σ^2. This is, in fact, a consistent estimator. (You can prove this using the consistency of $\hat{\beta}_0$ and $\hat{\beta}_1$, and continuity.) It is, however, a slightly biased estimator. Specifically

$$s^2 = \frac{n}{n-2}\hat{\sigma}^2 \tag{1.62}$$

is an unbiased estimator of σ^2, though one with a larger variance.

1.6.2 Residuals

The **residual** value at a data point is the difference between the actual value of the response y_i and the fitted value $\hat{m}(x_i)$:

$$e_i = y_i - \hat{m}(x_i) = y_i - (\hat{\beta}_0 + \hat{\beta}_1 x_i) \tag{1.63}$$

This may look like re-arranging the basic equation for the linear regression model,

$$\epsilon_i = Y_i - (\beta_0 + \beta_1 x_i) \tag{1.64}$$

and it *is* similar, but it's *not* the same. The right-hand side of Eq. 1.64 involves the true parameters. The right-hand side of Eq. 1.63 involves the *estimated* parameters, which are different. In particular, the estimated parameters are functions of *all* the noise variables. Hence, the residuals are not the noise terms: $e_i \neq \epsilon_i$.

Note also that

$$\frac{1}{n}\sum_{i=1}^{n} e_i = 0 \tag{1.65}$$

This is *reminiscent* of $\mathbb{E}[\epsilon] = 0$, but generally $n^{-1}\sum_{i=1}^{n}\epsilon_i \neq 0$. In fact, Eq. 1.65 implies that the residuals are actually linearly dependent on each other, while the ϵ_i are not.

Sum of squared errors The sum of squared errors for a fitted regression is :

$$SSE = \sum_{i=1}^{n} e_i^2 = \sum_{i=1}^{n} (y_i - \hat{m}(x_i))^2 = \sum_{i=1}^{n} (y_i - (\hat{\beta}_0 + \hat{\beta}_1 x_i))^2 \tag{1.66}$$

Despite these differences, there is enough of a relationship between the ϵ_i and the e_i that a lot of our model-checking and diagnostics will be done in terms of the residuals. You should get used to looking at them for basically any model you estimate, or even think seriously about estimating.

1.7 Gaussian distribution

We have, so far, imposed very mild assumptions on the noise term ϵ. The advantage of this is that our results apply to many different situations. The drawback is that these results are not that precise. If we made more detailed assumptions about ϵ, we could make more precise inferences.

There are lots of forms of distributions for ϵ which we might consider, and which are compatible with the assumptions of the simple linear regression model. The most popular distribution is the Gaussian distribution (Figure 1.2)

The result is the Gaussian-noise simple linear regression model:

1. The distribution of X is arbitrary (and perhaps X is even non-random).
2. If $X = x$, then $Y = \beta_0 + \beta_1 x + \epsilon$, for some constants ("coefficients", "parameters") β_0 and β_1, and some random noise variable ϵ.
3. $\epsilon \sim N(0, \sigma^2)$, independent of X.
4. ϵ is independent across observations.

You will notice that these assumptions are strictly stronger than those of the simple linear regression model. More exactly, the first two assumptions are the same, while the third and fourth assumptions of the Gaussian-noise model imply the corresponding assumptions of the other model. This means that everything

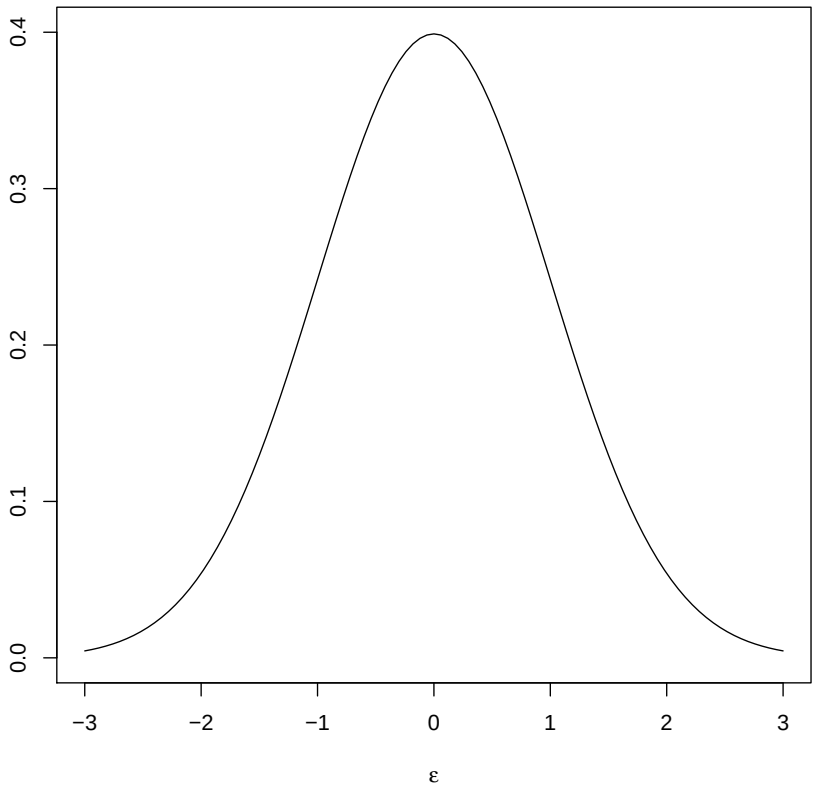


FIGURE 1.2: *The density of the standard Gaussian distribution admits mean* $\mathbb{E}[\epsilon] = 0$ *and variance* $\mathrm{Var}(\epsilon) = 1$.

we have done so far directly applies to the Gaussian-noise model. On the other hand, the stronger assumptions let us say more. They tell us, exactly, the probability distribution for Y given X, and so will let us get exact distributions for predictions and for other inferential statistics.

Why the Gaussian noise model? Why should we think that the noise around the regression line would follow a Gaussian (or normal) distribution, independent of X? There are two big reasons.

1. *Central limit theorem* The noise might be due to adding up the effects of lots of little random causes, all nearly independent of each other and of X, where each of the effects are of roughly similar magnitude. Then the central limit theorem will take over, and the distribution of the sum of effects will indeed be very close to Gaussian. For Gauss's original context, X was (simplifying) "Where is such-and-such-a-planet in space?", Y was "Where does an astronomer record the planet as appearing in the sky?", and noise came from defects in the telescope, eye-twitches, atmospheric distortions, etc., etc., so this was pretty reasonable.
2. *Mathematical convenience* Assuming Gaussian noise lets us work out a very complete theory of inference and prediction for the model. Getting precise inference results without the Gaussian-noise assumption is usually far more complicated.

1.7.1 Maximum Likelihood Estimation

The **likelihood** of a parameter value on a data set is the probability density of the data under those parameters. We could not work with the likelihood with the simple linear regression model, because we did not specify the noise-distribution to compute a density. With the Gaussian-noise model, however, we can write down the conditional likelihood given X. For the Gaussian simple linear model, the parameters are b_0, b_1, σ^2, then $Y|X = x \sim N(b_0 + b_1 x, \sigma^2)$, and Y_i and Y_j are independent given X_i and X_j, so the over-all likelihood is

$$\prod_{i=1}^{n} \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(y_i-(b_0+b_1 x_i))^2}{2\sigma^2}} \tag{1.67}$$

As usual, we work with the log-likelihood, which gives us the same information but replaces products with sums:

$$L(b_0, b_1, \sigma^2) = -\frac{n}{2}\log 2\pi - \frac{n}{2}\log \sigma^2 - \frac{1}{2\sigma^2}\sum_{i=1}^{n}(y_i - (b_0 + b_1 x_i))^2 \tag{1.68}$$

We recall from mathematical statistics that when we've got a likelihood function, we generally want to maximize it. That is, we want to find the parameter values which make the data we observed as likely as the model will allow. Solving this simple maximization problem can be done by looking for critical points and then check which of these critical points are maximizers.

$$\frac{\partial L}{\partial b_0} = -\frac{1}{2\sigma^2}\sum_{i=1}^{n} 2(y_i - (b_0 + b_1 x_i))(-1) \tag{1.69}$$

$$\frac{\partial L}{\partial b_1} = -\frac{1}{2\sigma^2}\sum_{i=1}^{n} 2(y_i - (b_0 + b_1 x_i))(-x_i) \tag{1.70}$$

Notice that when we set these derivatives to zero, all the multiplicative constants can be neglected. The above display simplifies to

$$\sum_{i=1}^{n} y_i - (\widehat{\beta_0} + \widehat{\beta_1} x_i) = 0 \tag{1.71}$$

$$\sum_{i=1}^{n} (y_i - (\widehat{\beta_0} + \widehat{\beta_1} x_i)) x_i = 0 \tag{1.72}$$

We recognize, up to a factor of $1/n$, the equations we got from the method of least squares (Eq. 1.12). That means that the least squares solution *is* the maximum likelihood estimate under the Gaussian noise model.

Now let's take the derivative with respect to σ^2:

$$\frac{\partial L}{\partial \sigma^2} = -\frac{n}{\sigma^2} + \frac{1}{2(\sigma^2)^2}\sum_{i=1}^{n} (y_i - (b_0 + b_1 x_i))^2 \tag{1.73}$$

Setting this to 0 at the optimum, including multiplying through by $\widehat{\sigma}^4$, we get

$$\widehat{\sigma^2} = \frac{1}{n}\sum_{i=1}^{n} (y_i - (\widehat{\beta_0} + \widehat{\beta_1} x_i))^2 \tag{1.74}$$

Notice that the right-hand side is just the sample mean squared error.

Maximum likelihood estimates of the regression curve coincide with least-squares estimates when the noise around the curve is additive, Gaussian, of constant variance, and both independent of X and of other noise terms. These were all assumptions we used in setting up a log-likelihood which was, up to constants, proportional to the (negative) mean-squared error.

1.8 Least-Squares in R

The basic command for fitting a linear model by least squares in R is `lm`. It has a huge number of options, and can do a lot more than we will ask it to here, but for our purposes we use it as follows:

```
lm(y ~ x, data = toydata)
```

Here `toydata` is a data frame containing the data we want to fit a regression to, and the first part, the **formula**, tells `lm` that we want to regress the column

of `df` called `y` on the column called `x`. By default, `lm` always fits its regression models by least squares.

What `lm` returns is a complete object containing not only the estimates for the intercept and the slope but also several statistical tests. If you just print it out, it seems to be only the intercept and the slope:

```
sim.linmod <- function(n, beta.0, beta.1, width, df) {
    x <- runif(n, min = -width/2, max = width/2)
    epsilon <- rt(n, df = df)
    y <- beta.0 + beta.1 * x + epsilon
    return(data.frame(x = x, y = y))
}
toy.data <- sim.linmod(n = 10, beta.0 = 5, beta.1 = -2, width = 4, df = 3)
lm(y ~ x, data = toy.data)
##
## Call:
## lm(formula = y ~ x, data = toy.data)
##
## Coefficients:
## (Intercept)            x
##       4.911       -2.150
```

In fact, `lm` has done *lots* of calculations as part of fitting the model, and stored many of the results into the object it returns; R just doesn't print all of that, unless you make it. We can see some of what's in there, though:

```
names(lm(y ~ x, data = toy.data))
##  [1] "coefficients"  "residuals"     "effects"       "rank"
##  [5] "fitted.values" "assign"        "qr"            "df.residual"
##  [9] "xlevels"       "call"          "terms"         "model"
```

(See `help(lm)`, under "Value", for the gory details.) It's annoying (and slow and error-prone) to keep having R re-estimate the model every time we want to refer back to it, so we usually store the estimated model under a new variable name:

```
toy.lm <- lm(y ~ x, data = toy.data)
```

We can access some of what's in the `lm` object by using special functions. A couple in particular will become close and familiar friends. `coefficients` gives us the vector of coefficients:

```
coefficients(toy.lm)
## (Intercept)           x
##    4.910538   -2.149905
```

`fitted` gives us the vector of fitted values, in the order which the data points appeared:

```
fitted(toy.lm)
##        1        2        3        4        5        6        7        8
## 2.210722 5.916679 6.066816 4.923140 6.383513 8.083748 8.533207 1.090057
##        9       10
## 5.136045 1.235011
```

`residuals` gives us the vector of residuals (ditto order):

```
residuals(toy.lm)
##          1          2          3          4          5          6
##  0.3727931 -0.2877128 -0.1279798 -0.3209172  0.5900863 -0.3675145
##          7          8          9         10
##  0.8998177 -0.9962088 -1.1400326  1.3776686
```

(How would you use `residuals` to calculate s^2? To calculate $\frac{n}{n-2}s^2$?)

You might think that `plot(toy.lm)` would draw a picture of the fitted model; instead, it goes through a bunch of diagnostic plots, which we will get to later. If we want to add a line based on the model to an existing plot, we use `abline`, as in Figure 1.3.

`fitted` gives us the model's predictions at the particular x_i where we happened to have data. In principle, though, the model has an opinion about what $\mathbb{E}[Y|X = x]$ should be at every possible value of x. To extract that, we use a function called `predict`. Naturally enough, we need to tell it both which model we want to use (since we could have more than one floating around), *and* where to make the predictions:

```
predict(object, newdata)
```

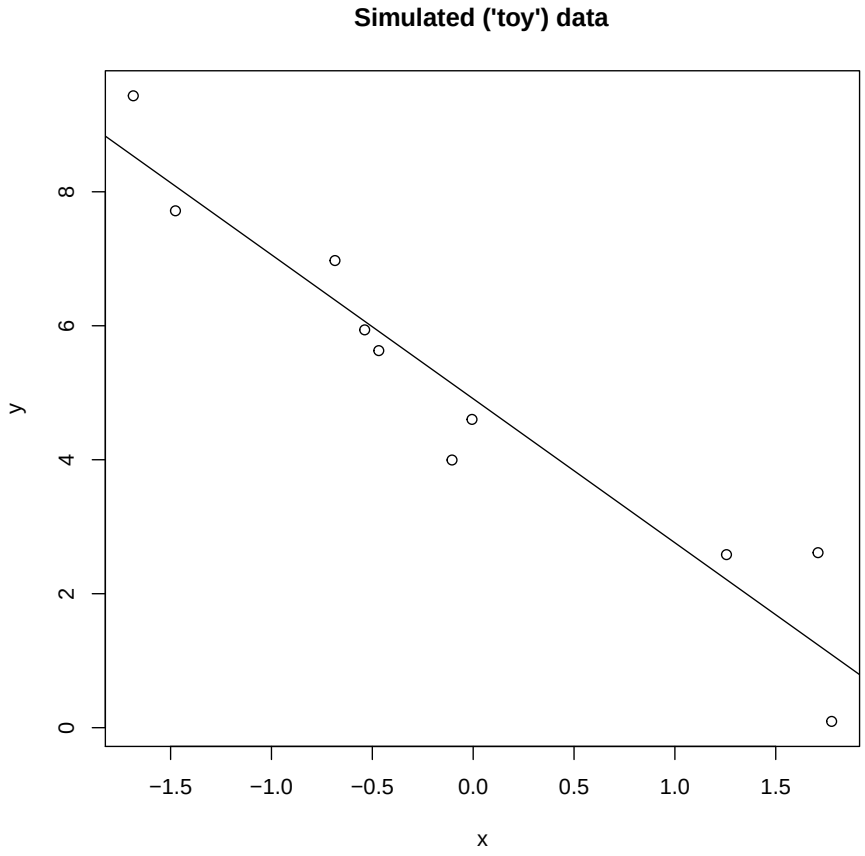
Here the first argument, what `predict` somewhat obscurely calls `object`, is the estimated regression model, like our `toy.lm`. (It is *not* the name of the estimating function, like `lm`.) The second argument, `newdata`, is a data frame with a column whose name matches the column name of the predictor variable in our original data frame. Thus

```
predict(toy.lm, newdata = data.frame(x = 1:5))
##          1          2          3          4          5
##  2.7606327  0.6107273 -1.5391781 -3.6890835 -5.8389889
```

gives us the fitted model's predictions at the integers from 1 to 5.

You might well think that if `newdata` were missing, then `predict` would throw an error. You might very well think that.

```
predict(toy.lm)
##        1        2        3        4        5        6        7        8
## 2.210722 5.916679 6.066816 4.923140 6.383513 8.083748 8.533207 1.090057
##        9       10
## 5.136045 1.235011
predict(toy.lm, data = data.frame(x = 1:5))
##        1        2        3        4        5        6        7        8
```




```
plot(y ~ x, data = toy.data, xlab = "x", ylab = "y", main = "Simulated ('toy') data")
abline(toy.lm)
```

FIGURE 1.3: *Using* `abline` *to add the line of an estimated linear regression model to a plot.*

```
## 2.210722 5.916679 6.066816 4.923140 6.383513 8.083748 8.533207 1.090057
##        9       10
## 5.136045 1.235011
```

For reasons best known to the designers of R, when `newdata` is missing or mal-formed, `predict` returns the fitted values on the original data. On the other hand, you *will* get an error if `newdata` exists but doesn't contain the right column name:

```
predict(toy.lm, newdata = data.frame(zebra = 1:5))
```

```
## Error in eval(expr, envir, enclos):  object 'x' not found
```

Extraneous columns, however, are just ignored:

```
predict(toy.lm, newdata = data.frame(x = 1:5, zebra = 6:10))
##          1          2          3          4          5
##  2.7606327  0.6107273 -1.5391781 -3.6890835 -5.8389889
```

There is one further option to `predict` which is worth mentioning at this time. If we set `se.fit=TRUE`, we get the standard errors of the fitted values, i.e., the square roots of the variances:

```
predict(toy.lm, newdata = data.frame(x = 1:5), se.fit = TRUE)
## $fit
##          1          2          3          4          5
##  2.7606327  0.6107273 -1.5391781 -3.6890835 -5.8389889
##
## $se.fit
##         1         2         3         4         5
## 0.3584318 0.5391797 0.7480319 0.9669459 1.1903834
##
## $df
## [1] 8
##
## $residual.scale
## [1] 0.8532392
```

Notice that what this gives us back is not a vector but a *list*, whose first two entries are vectors. (We will come back to what the `df` means, but you should already be able to guess what `residual.scale` might be.)

Chapter 2

Inference in the Gaussian-Noise Simple Linear Regression Model

In the previous chapter, we came up with **point estimators** of the parameters and the conditional mean (prediction) function. In this chapter we will focus on the statistical properties of these point estimates. This includes building confidence intervals for the parameters and testing hypotheses about them.

To accomplish all this, we first need to understand the sampling distribution of our point estimators. We can find them, mathematically, but they involve the unknown true parameters in inconvenient ways. We will therefore work to find combinations of our estimators and the true parameters with fixed, parameter-free distributions; we'll get our confidence sets and our hypothesis tests from them.

Throughout this chapter, I am assuming, unless otherwise noted, that all of the assumptions of the Gaussian-noise simple linear regression model hold.

2.1 Sampling Distribution of $\hat{\beta}_0$, $\hat{\beta}_1$ and $\hat{\sigma}^2$

The Gaussian-noise simple linear regression model has three parameters: the intercept β_0, the slope β_1, and the noise variance σ^2. We've seen, previously, how to estimate all of these by maximum likelihood; the MLE for the βs is the

same as their least-squares estimates. These are

$$\hat{\beta}_1 = \frac{c_{XY}}{s_X^2} = \sum_{i=1}^{n} \frac{x_i - \overline{x}}{n s_X^2} y_i \tag{2.1}$$

$$\hat{\beta}_0 = \overline{y} - \hat{\beta}_1 \overline{x} \tag{2.2}$$

$$\hat{\sigma}^2 = \frac{1}{n} \sum_{i=1}^{n} (y_i - (\hat{\beta}_0 + \hat{\beta}_1 x_i))^2 \tag{2.3}$$

We have also seen how to re-write the first two of these as a deterministic part plus a weighted sum of the noise terms ϵ:

$$\hat{\beta}_1 = \beta_1 + \sum_{i=1}^{n} \frac{x_i - \overline{x}}{n s_X^2} \epsilon_i \tag{2.4}$$

$$\hat{\beta}_0 = \beta_0 + \frac{1}{n} \sum_{i=1}^{n} \left(1 - \overline{x} \frac{x_i - \overline{x}}{s_X^2}\right) \epsilon_i \tag{2.5}$$

Finally, we have our modeling assumption that the ϵ_i are independent Gaussians, $\epsilon_i \sim N(0, \sigma^2)$.

2.1.1 Reminders of Basic Properties of Gaussian Distributions

Suppose $U \sim N(\mu, \sigma^2)$. By the basic algebra of expectations and variances, $\mathbb{E}[a + bU] = a + b\mu$, while $\mathrm{Var}[a + bU] = b^2\sigma^2$. This would be true of any random variable; a special property of Gaussians is that $a + bU \sim N(a + b\mu, b^2\sigma^2)$.

Suppose $U_1, U_2, \ldots U_n$ are *independent* Gaussians, with means μ_i and variances σ_i^2. Then

$$\sum_{i=1}^{n} U_i \sim N(\sum_i \mu_i, \sum_i \sigma_i^2)$$

That the expected values add up for a sum is true of all random variables; that the variances add up is true for all uncorrelated random variables. That the sum follows the same type of distribution as the summands is a special property of Gaussians.

2.1.2 Sampling Distribution of $\hat{\beta}_1$

Since we're assuming Gaussian noise, the ϵ_i are independent Gaussians, $\epsilon_i \sim N(0, \sigma^2)$. Hence (using the first basic property of Gaussians)

$$\frac{x_i - \overline{x}}{n s_X^2} \epsilon_i \sim N(0, \left(\frac{x_i - \overline{x}}{n s_X^2}\right)^2 \sigma^2)$$

```
sim.gnslrm <- function(x, intercept, slope, sigma.sq, coefficients = TRUE) {
    n <- length(x)
    y <- intercept + slope * x + rnorm(n, mean = 0, sd = sqrt(sigma.sq))
    if (coefficients) {
        return(coefficients(lm(y ~ x)))
    }
    else {
        return(data.frame(x = x, y = y))
    }
}
x <- seq(from = -5, to = 5, length.out = 42)
```

FIGURE 2.1: *Code setting up a simulation of a Gaussian-noise simple linear regression model, along a fixed vector of x_i values.*

Thus, using the second basic property of Gaussians,

$$\sum_{i=1}^{n} \frac{x_i - \overline{x}}{n s_X^2} \epsilon_i \sim N(0, \sigma^2 \sum_{i=1}^{n} \left(\frac{x_i - \overline{x}}{n s_X^2} \right)^2) \tag{2.6}$$

$$= N(0, \frac{\sigma^2}{n s_X^2}) \tag{2.7}$$

Using the first property of Gaussians again,

$$\hat{\beta}_1 \sim N(\beta_1, \frac{\sigma^2}{n s_X^2}) \tag{2.8}$$

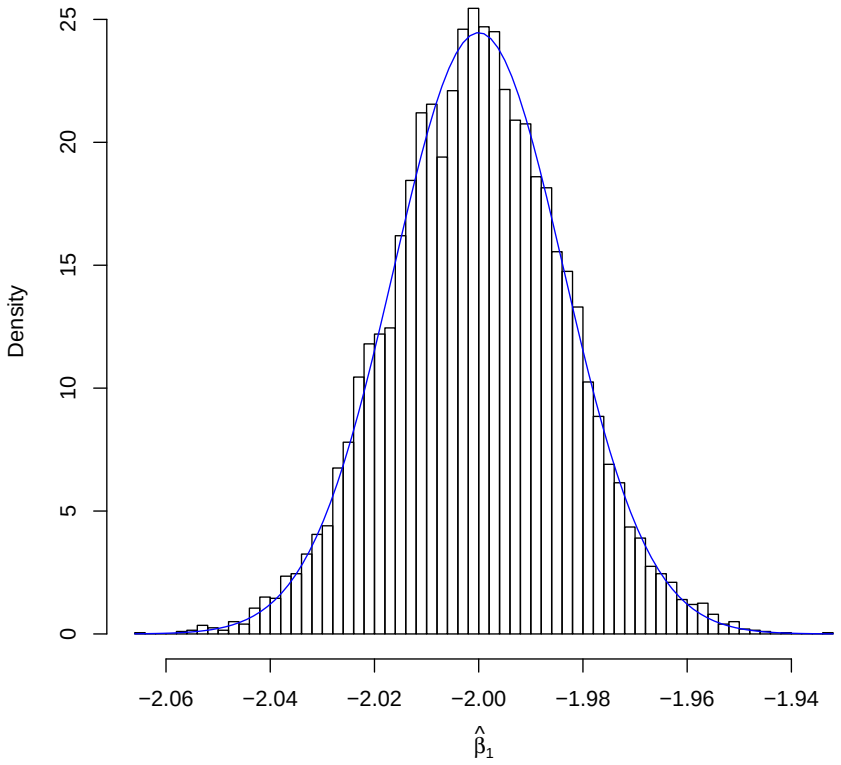
This is the distribution of estimates we'd see if we repeated the experiment (survey, observation, etc.) many times, and collected the results. Every particular run of the experiment would give a slightly different $\hat{\beta}_1$, but they'd average out to β_1, the average squared difference from β_1 would be $\sigma^2/n s_X^2$, and a histogram of them would follow the Gaussian probability density function (Figure 2.2).

It is a bit hard to use Eq. 2.8, because it involves two of the unknown parameters. We can manipulate it a bit to remove one of the parameters from the probability distribution,

$$\hat{\beta}_1 - \beta_1 \sim N(0, \frac{\sigma^2}{n s_X^2})$$

but that still has σ^2 on the right hand side. Standardizing the above statistic, we obtain that

$$\frac{\hat{\beta}_1 - \beta_1}{\sigma^2/\sqrt{n s_X^2}} \sim N(0,1).$$




```
many.coefs <- replicate(10000, sim.gnslrm(x = x, 5, -2, 0.1, coefficients = TRUE))
hist(many.coefs[2, ], breaks = 50, freq = FALSE, xlab = expression(hat(beta)[1]),
    main = "")
theoretical.se <- sqrt(0.1/(length(x) * var(x)))
curve(dnorm(x, mean = -2, sd = theoretical.se), add = TRUE, col = "blue")
```

FIGURE 2.2: *Simulating 10,000 runs of a Gaussian-noise simple linear regression model, calculating $\hat{\beta}_1$ each time, and comparing the histogram of estimates to the theoretical Gaussian distribution (Eq. 2.8, in blue).*

2.1.3 Sampling Distribution of $\hat{\beta}_0$

Starting from Eq. 2.5 rather than Eq. 2.4, an argument exactly parallel to the one we just went through gives

$$\hat{\beta}_0 \sim N(\beta_0, \frac{\sigma^2}{n}\left(1 + \frac{\overline{x}^2}{s_X^2}\right))$$

It follows, again by parallel reasoning, that

$$\frac{\hat{\beta}_0 - \beta_0}{\sqrt{\frac{\sigma^2}{n}\left(1 + \frac{\overline{x}^2}{s_X^2}\right)}} \sim N(0,1)$$

The right-hand side of this equation is admirably simple and easy for us to calculate, but the left-hand side unfortunately involves two unknown parameters, and that complicates any attempt to use it.

2.1.4 Sampling Distribution of $\hat{\sigma}^2$

Recall that

$$\hat{\sigma}^2 = \frac{1}{n}\sum_{i=1}^{n} e_i^2.$$

When $\epsilon_i \sim N(0, \sigma^2)$, it can be shown that (Exercise)

$$\frac{n\hat{\sigma}^2}{\sigma^2} \sim \chi^2_{n-2}.$$

Note that this result lets us calculate the probability that $\hat{\sigma}^2$ is within any given *factor* of σ^2. Let give some justification for the above assertion.

χ^2 distribution If $Z \sim N(0,1)$, then $Z^2 \sim \chi^2_1$ *by definition* (of the χ^2_1 distribution). From this, it follows that $\mathbb{E}\left[\chi^2_1\right] = 1$, $\mathrm{Var}\left[\chi^2_1\right] = \mathbb{E}\left[Z^4\right] - (\mathbb{E}\left[Z^2\right])^2 = 2$. If $Z_1, Z_2, \ldots Z_d \sim N(0,1)$ and are independent, then the χ^2_d distribution is *defined* to be the distribution of $\sum_{i=1}^{d} Z_i^2$. By simple algebra, it follows that $\mathbb{E}\left[\chi^2_d\right] = d$ while $\mathrm{Var}\left[\chi^2_d\right] = 2d$.

Back to the sum of squared noise terms ϵ_i isn't a standard Gaussian, but ϵ_i/σ is, so

$$\frac{\sum_{i=1}^{n}\epsilon_i^2}{\sigma^2} = \sum_{i=1}^{n}(\frac{\epsilon_i}{\sigma})^2 \sim \chi^2_n$$

The term $n\frac{\hat{\sigma}^2}{\sigma^2} = \sum_{i=1}^{n} e_i^2$ follows a similar distribution with n replaced by $n-2$. There is a geometric interpretation for this difference. The sum of squared errors, $\sum_{i=1}^{n} e_i^2$, is the squared length of the n-dimensional vector of residuals, $(e_1, e_2, \ldots e_n)$. But the residuals must obey the two equations $\sum_i e_i = 0$,

$\sum_i x_i e_i = 0$, so the residual vector actually is confined to an $(n-2)$-dimensional linear subspace. Thus we only end up adding up $(n-2)$ *independent* contributions to its length. If we estimated more parameters, we'd have more estimating equations, and so more constraints on the vector of residuals.

2.1.5 Standard Errors of $\hat{\beta}_0$ and $\hat{\beta}_1$

The **standard error** of an estimator is its standard deviation. We've just seen that the true standard errors of $\hat{\beta}_0$ and $\hat{\beta}_1$ are, respectively,

$$\text{se}\left(\hat{\beta}_1\right) = \frac{\sigma}{s_x\sqrt{n}} \tag{2.9}$$

$$\text{se}\left(\hat{\beta}_0\right) = \frac{\sigma}{\sqrt{n}s_X}\sqrt{s_X^2 + \overline{x}^2} \tag{2.10}$$

Unfortunately, these standard errors involve the unknown parameter σ^2 (or its square root σ, equally unknown to us).

We can, however, *estimate* the standard errors. The maximum-likelihood estimates just substitute $\hat{\sigma}$ for σ:

$$\hat{\text{se}}\left(\hat{\beta}_1\right) = \frac{\hat{\sigma}}{s_x\sqrt{n}} \tag{2.11}$$

$$\hat{\text{se}}\left(\hat{\beta}_0\right) = \frac{\hat{\sigma}}{s_X\sqrt{n}}\sqrt{s_X^2 + \overline{x}^2} \tag{2.12}$$

For later theoretical purposes, however, things will work out slightly nicer if we use the de-biased version, $\frac{n}{n-2}\hat{\sigma}^2$:

$$\hat{\text{se}}\left(\hat{\beta}_1\right) = \frac{\hat{\sigma}}{s_x\sqrt{n-2}} \tag{2.13}$$

$$\hat{\text{se}}\left(\hat{\beta}_0\right) = \frac{\hat{\sigma}}{s_x\sqrt{n-2}}\sqrt{s_X^2 + \overline{x}^2} \tag{2.14}$$

These standard errors — approximate or estimated though they be — are one important way of quantifying how much uncertainty there is around our point estimates. However, we can't use them, *alone* to say anything terribly precise about, say, the probability that β_1 is in the interval $[\hat{\beta}_1 - \hat{\text{se}}\left(\hat{\beta}_1\right), \hat{\beta}_1 - \hat{\text{se}}\left(\hat{\beta}_1\right)]$, which is the sort of thing we'd want to be able to give guarantees about the reliability of our estimates.

2.2 Sampling distribution of $(\hat{\beta} - \beta)/\hat{\text{se}}\left(\hat{\beta}\right)$

Recall that

$$\frac{\hat{\beta}_1 - \beta_1}{\text{se}\left(\hat{\beta}_1\right)} \sim N(0,1)$$

If the Oracle told us σ^2, we'd know $\text{se}\left(\hat{\beta}_1\right)$, and so we could assert that (for example)

$$\mathbb{P}\left(-1.96 \leq \frac{\hat{\beta}_1 - \beta_1}{\text{se}\left(\hat{\beta}_1\right)} \leq 1.96\right) \tag{2.15}$$

$$= \Phi(1.96) - \Phi(-1.96) = 0.95 \tag{2.16}$$

where Φ is the cumulative distribution function of the $N(0,1)$ distribution.

When we do not know σ^2, we can use the following fact:

Definition 1. *Let $Z \sim N(0,1)$, $S^2 \sim \chi^2_d$, and Z and S^2 are independent. The distributiion of $\frac{Z}{\sqrt{S^2/d}}$ is called Student t-distribution with d degrees of freedom. We use the notation:*

$$\frac{Z}{\sqrt{S^2/d}} \sim t_d$$

We have

$$\frac{\hat{\beta}_1 - \beta_1}{\hat{\text{se}}\left(\hat{\beta}_1\right)} = \frac{\hat{\beta}_1 - \beta_1}{\sigma} \frac{\sigma}{\hat{\text{se}}\left(\hat{\beta}_1\right)} \tag{2.17}$$

$$= \frac{\frac{\hat{\beta}_1 - \beta_1}{\sigma}}{\frac{\hat{\text{se}}\left(\hat{\beta}_1\right)}{\sigma}} \tag{2.18}$$

$$\stackrel{d}{=} \frac{N(0, 1/ns_X^2)}{\frac{\hat{\sigma}}{s_x \sigma\sqrt{n-2}}} \tag{2.19}$$

$$\stackrel{d}{=} \frac{s_X N(0, 1/ns_X^2)}{\frac{\hat{\sigma}}{\sigma\sqrt{n-2}}} \tag{2.20}$$

$$\stackrel{d}{=} \frac{N(0, 1/n)}{\frac{\hat{\sigma}}{\sigma\sqrt{n-2}}} \tag{2.21}$$

$$\stackrel{d}{=} \frac{\sqrt{n}N(0, 1/n)}{\frac{\sqrt{n}\hat{\sigma}}{\sigma\sqrt{n-2}}} \tag{2.22}$$

$$\stackrel{d}{=} \frac{N(0,1)}{\sqrt{\frac{n\hat{\sigma}^2}{\sigma^2}\frac{1}{n-2}}} \tag{2.23}$$

$$\stackrel{d}{=} \frac{N(0,1)}{\sqrt{\chi^2_{n-2}/(n-2)}} \sim t_{n-2}. \tag{2.24}$$

To sum up:

Proposition 1. *Using the $\hat{\text{se}}\left(\hat{\beta}_1\right)$ of Eq. 2.13,*

$$\frac{\hat{\beta}_1 - \beta_1}{\hat{\text{se}}\left(\hat{\beta}_1\right)} \sim t_{n-2} \tag{2.25}$$

Note that $\hat{\text{se}}\left(\hat{\beta}_1\right)$ is a data-driven statistic. That means that we can compute it without knowing any of the true parameters which is an appealing property in applications.

By exactly parallel reasoning, we may also demonstrate that

$$\frac{\hat{\beta}_0 - \beta_0}{\hat{\text{se}}\left(\hat{\beta}_0\right)} \sim t_{n-2}$$

2.3 Building Confidence Intervals from Sampling distributions

Once we know how to calculate sampling intervals, we can plot the sampling interval for every possible value of β_1 (Figure 2.3). They're the region marked off by two parallel lines, one $k(n,\alpha)\hat{\text{se}}\left(\hat{\beta}_1\right)$ above the main diagonal and one equally far below the main diagonal.

The sampling intervals (as in Figure 2.3) are theoretical constructs — mathematical consequences of the assumptions in the the probability model that (we hope) describes the world. After we gather data, we can actually calculate $\hat{\beta}_1$. This is a random quantity, but it will have some particular value on any data set. We can mark this realized value, and draw a horizontal line across the graph at that height (Figure 2.4).

We define the **confidence set**, with **confidence level** $1-\alpha$, as

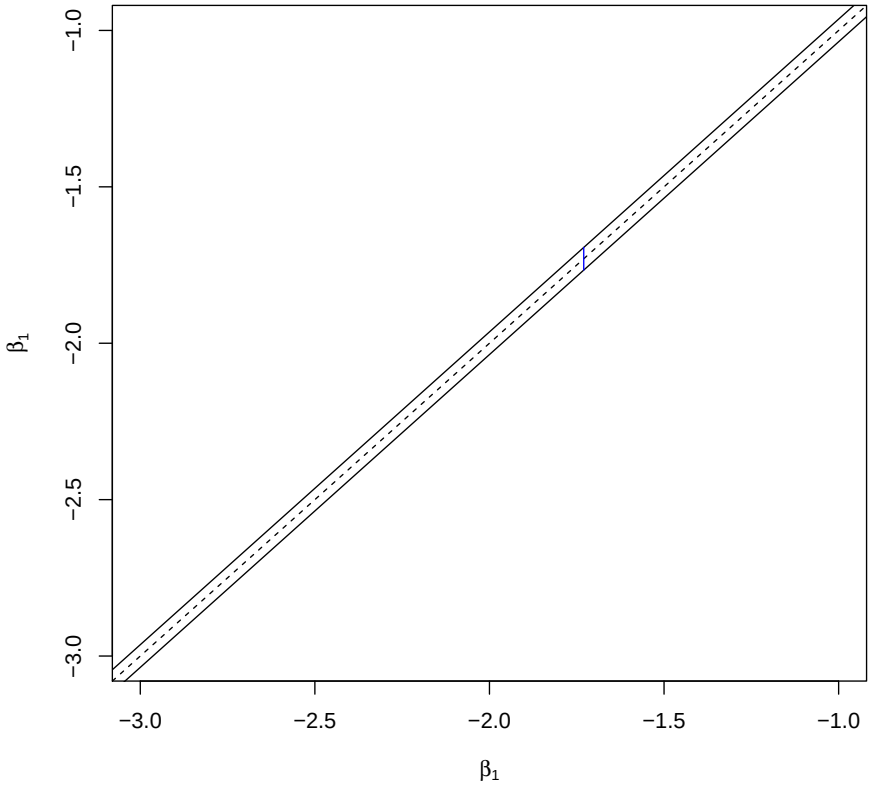
$$\Delta_1(\alpha) = [\hat{\beta}_1 - k(n,\alpha)\hat{\text{se}}\left(\hat{\beta}_1\right), \hat{\beta}_1 + k(n,\alpha)\hat{\text{se}}\left(\hat{\beta}_1\right)]. \tag{2.26}$$

We have indeed $\mathbb{P}\left(\beta_1 \in \Delta_1(\alpha)\right) = 1-\alpha$.

Width of the confidence interval Notice that the width of the confidence interval is $2k(n,\alpha)\hat{\text{se}}\left(\hat{\beta}_1\right)$. This tells us what controls the width of the confidence interval:

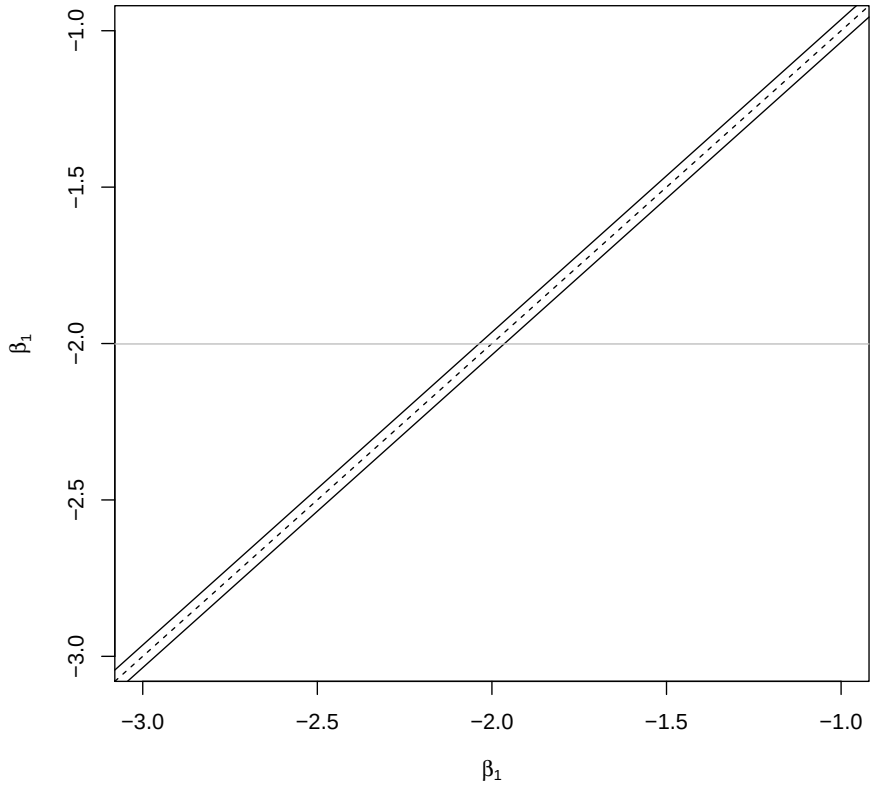
1. As α shrinks, the interval widens. (High confidence comes at the price of big margins of error.)
2. As n grows, the interval shrinks. (Large samples mean precise estimates.)
3. As σ^2 increases, the interval widens. (The more noise there is around the regression line, the less precisely we can measure the line.)
4. As s_X^2 grows, the interval shrinks. (Widely-spread measurements give us a precise estimate of the slope.)

Intercept β_0 By exactly parallel reasoning, a $1-\alpha$ confidence interval for β_0 is $[\hat{\beta}_0 - k(n,\alpha)\hat{\text{se}}\left(\hat{\beta}_0\right), \hat{\beta}_0 + k(n,\alpha)\hat{\text{se}}\left(\hat{\beta}_0\right)]$.



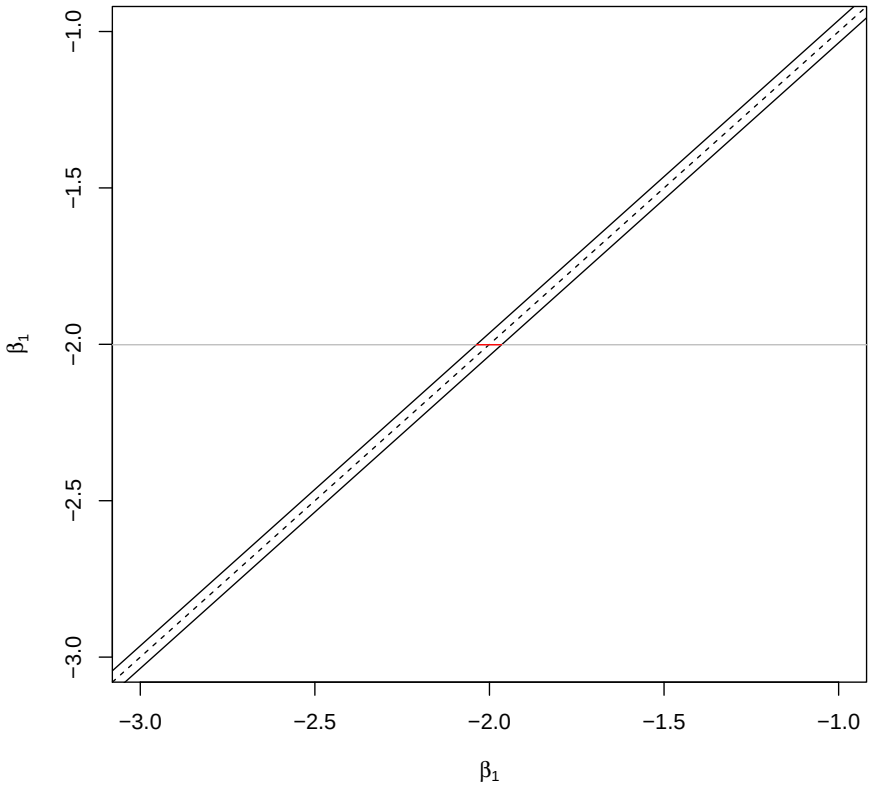

```
lm.sim <- lm(y ~ x, data = sim.gnslrm(x = x, 5, -2, 0.1, coefficients = FALSE))
hat.sigma.sq <- mean(residuals(lm.sim)^2)
se.hat.beta.1 <- sqrt(hat.sigma.sq/(var(x) * (length(x) - 2)))
alpha <- 0.02
k <- qt(1 - alpha/2, df = length(x) - 2)
plot(0, xlim = c(-3, -1), ylim = c(-3, -1), type = "n", xlab = expression(beta[1]),
    ylab = expression(hat(beta)[1]), main = "")
abline(a = k * se.hat.beta.1, b = 1)
abline(a = -k * se.hat.beta.1, b = 1)
abline(a = 0, b = 1, lty = "dashed")
beta.1.star <- -1.73
segments(x0 = beta.1.star, y0 = k * se.hat.beta.1 + beta.1.star, x1 = beta.1.star,
    y1 = -k * se.hat.beta.1 + beta.1.star, col = "blue")
```

FIGURE 2.3: *Sampling intervals for $\hat{\beta}_1$ as a function of β_1. For compatibility with the earlier simulation, I have set $n = 42$, $s_X^2 = 9$, and (from one run of the model) $\hat{\sigma}^2 = 0.08$; and, just because $\alpha = 0.05$ is cliched, $\alpha = 0.02$. Equally arbitrarily, the blue vertical line illustrates the sampling interval when $\beta_1 = -1.73$.*




```
plot(0, xlim = c(-3, -1), ylim = c(-3, -1), type = "n", xlab = expression(beta[1]),
    ylab = expression(hat(beta)[1]), main = "")
abline(a = k * se.hat.beta.1, b = 1)
abline(a = -k * se.hat.beta.1, b = 1)
abline(a = 0, b = 1, lty = "dashed")
beta.1.hat <- coefficients(lm.sim)[2]
abline(h = beta.1.hat, col = "grey")
```

FIGURE 2.4: *As in Figure 2.3, but with the addition of a horizontal line marking the observed value of $\hat{\beta}_1$ on a particular realization of the simulation (in grey).*




```
plot(0, xlim = c(-3, -1), ylim = c(-3, -1), type = "n", xlab = expression(beta[1]),
    ylab = expression(hat(beta)[1]), main = "")
abline(a = k * se.hat.beta.1, b = 1)
abline(a = -k * se.hat.beta.1, b = 1)
abline(a = 0, b = 1, lty = "dashed")
beta.1.hat <- coefficients(lm.sim)[2]
abline(h = beta.1.hat, col = "grey")
segments(x0 = beta.1.hat - k * se.hat.beta.1, y0 = beta.1.hat, x1 = beta.1.hat +
    k * se.hat.beta.1, y1 = beta.1.hat, col = "red")
```

FIGURE 2.5: *As in Figure 2.4, but with the* **confidence set** *marked in red. This is the collection of all β_1 where $\hat{\beta}_1$ falls within the $1 - \alpha$ sampling interval.*

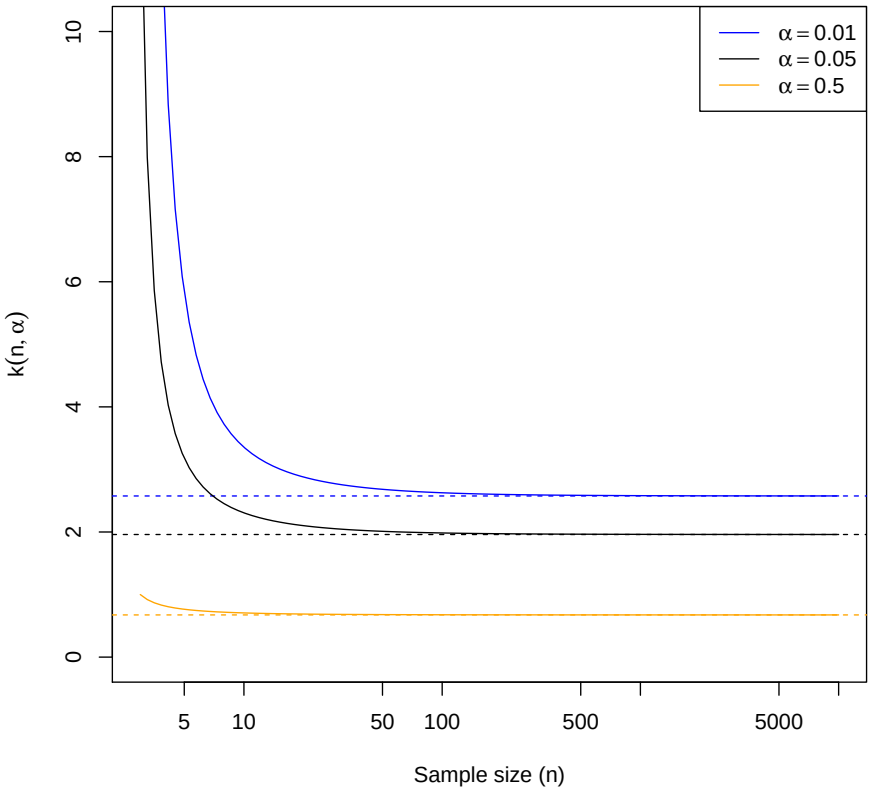
What α should we use? It's become conventional to set $\alpha = 0.05$. But there is nothing which stops you from taking a different value for α.

What about power? The **coverage** of a confidence set is the probability that it includes the true parameter value. This is not, however, the only virtue we want in a confidence set; if it was, we could just say "Every possible parameter is in the set", and have 100% coverage no matter what. We would also like the *wrong* values of the parameter to have a high probability of *not* being in the set. Just as the coverage is controlled by the size / false-alarm probability / type-I error rate α of the hypothesis test, the probability of excluding the wrong parameters is controlled by the power / miss probability / type-II error rate. Test with higher power exclude (correctly) more parameter values, and give smaller confidence sets.

Large-n Asymptotics As $n \to \infty$, $\hat{\sigma}^2 \to \sigma^2$. It follows (by continuity) that $\hat{\text{se}}\left(\hat{\beta}\right) \to \text{se}\left(\hat{\beta}\right)$. Hence,

$$\frac{\hat{\beta} - \beta}{\hat{\text{se}}\left(\hat{\beta}\right)} \to N(0,1)$$

which considerably simplifies the sampling intervals and confidence sets; as n grows, we can forget about the t distribution and just use the standard Gaussian distribution. Figure 2.6 plots the convergence of $k(n,\alpha)$ towards the $k(\infty,\alpha)$ we'd get from the Gaussian approximation. As you can see from the figure, by the time $n = 100$ —a quite small data set by modern standards — the difference between the t distribution and the standard-Gaussian is pretty trivial.




```
curve(qt(0.995, df = x - 2), from = 3, to = 10000, log = "x", ylim = c(0, 10),
    xlab = "Sample size (n)", ylab = expression(k(n, alpha)), col = "blue")
abline(h = qnorm(0.995), lty = "dashed", col = "blue")
curve(qt(0.975, df = x - 2), add = TRUE)
abline(h = qnorm(0.975), lty = "dashed")
curve(qt(0.75, df = x - 2), add = TRUE, col = "orange")
abline(h = qnorm(0.75), lty = "dashed", col = "orange")
legend("topright", legend = c(expression(alpha == 0.01), expression(alpha ==
    0.05), expression(alpha == 0.5)), col = c("blue", "black", "orange"), lty = "solid")
```

FIGURE 2.6: *Convergence of* $k(n, \alpha)$ *as* $n \to \infty$*, illustrated for* $\alpha = 0.01$*,* $\alpha = 0.05$ *and* $\alpha = 0.5$*. (Why do I plot the* 97.5^{th} *percentile when I'm interested in* $\alpha = 0.05$*?)*

2.4 Statistical Significance

2.4.1 p-Values

The test statistic for the Wald test,

$$T = \frac{\hat{\beta}_1 - \beta_1^*}{\hat{\text{se}}\left(\hat{\beta}_1\right)}$$

has the nice, intuitive property that it ought to be close to zero when the null hypothesis $\beta_1 = \beta_1^*$ is true, and take large values (either positive or negative) when the null hypothesis is false. When a test statistic works like this, it makes sense to summarize just how bad the data looks for the null hypothesis in a **p-value**: when our observed value of the test statistic is T_{obs}, the p-value is

$$P = \mathbb{P}\left(|T| \geq |T_{obs}|\right)$$

calculating the probability under the null hypothesis. (I write a capital P here as a reminder that this is a random quantity, though it's conventional to write the phrase "p-value" with a lower-case p.) This is the probability, under the null, of getting results which are at least as extreme as what we saw. It should be easy to convince yourself that rejecting the null in a level-α test is the same as getting a p-value $< \alpha$.

2.4.2 Statistical Significance

If we test the hypothesis that $\beta_1 = \beta_1^*$ and reject it, we say that the difference between β_1 and β_1^* is **statistically significant**. The hypothesis $\beta_1 = 0$ is particularly used in applications. When we reject the hypothesis that $\beta_1 = 0$, it means that it is really implausibly hard to fit this data with a flat line, as opposed to one with a slope. Conversely, when we accept the null hypothesis, it means that the difference between β_1 and 0 is not statistically significant but it does not implies that $\beta_1 = 0$. To see why, it is enough to realize that there are (at least) two reasons why our hypothesis test might retain the null $\beta_1 = 0$:

1. β_1 is, in fact, zero,
2. $\beta_1 \neq 0$, but $\hat{\text{se}}\left(\hat{\beta}_1\right)$ is so large that we can't tell anything about β_1 with any confidence.

Even a huge β_1 can be statistically insignificant, so long as $\hat{\text{se}}\left(\hat{\beta}_1\right)$ is large enough. Conversely, any β_1 which isn't exactly zero, no matter how close it might be to 0, will become statistically significant at any threshold once $\hat{\text{se}}\left(\hat{\beta}_1\right)$ is small enough. Since, as $n \to \infty$,

$$\hat{\text{se}}\left(\hat{\beta}_1\right) \to \frac{\sigma}{s_X\sqrt{n}}$$

we can show that $\hat{\text{se}}\left(\hat{\beta}_1\right) \to 0$, and $\frac{\hat{\beta}_1}{\hat{\text{se}}\left(\hat{\beta}_1\right)} \to \pm\infty$, unless β_1 is exactly 0.

2.5 Effect of Sample Size on Testing Significance of any Non-Zero Parameter

We investigate the asymptotic behavior of the test statistic when $n \to \infty$, and so at what happens to the p-value.Throughout, we will be testing the null hypothesis that $\beta_1 = 0$ (the analysis carries over to any arbitrary value of β_1 and similarly, it carries over with straightforward changes to testing hypotheses about the intercept β_0 too.)

We know that $\hat{\beta}_1 \sim N(\beta_1, \sigma^2/ns_X^2)$. This means

$$\begin{aligned} \hat{\beta}_1 &\sim \beta_1 + N(0, \sigma^2/ns_X^2) && (2.27)\\ &= \beta_1 + \frac{\sigma}{s_X\sqrt{n}} N(0,1) && (2.28)\\ &= \beta_1 + O(1/\sqrt{n}) && (2.29) \end{aligned}$$

where $O(f(n))$ is read "order-of $f(n)$", meaning that it's a term whose size grows like $f(n)$ as $n \to \infty$, and we don't want (or need) to keep track of the details. Similarly, since $n\hat{sigma}^2/\sigma^2 \sim \chi^2_{n-2}$, we have

$$\begin{aligned} n\hat{\sigma}^2 &\sim \sigma^2 \chi^2_{n-2} && (2.30)\\ \hat{\sigma}^2 &\sim \sigma^2 \frac{\chi^2_{n-2}}{n} && (2.31) \end{aligned}$$

Since $\mathbb{E}\left[\chi^2_{n-2}\right] = n-2$ and $\mathrm{Var}\left[\chi^2_{n-2}\right] = 2(n-2)$,

$$\begin{aligned} \mathbb{E}\left[\frac{\chi^2_{n-2}}{n}\right] &= \frac{n-2}{n} \to 1 && (2.32)\\ \mathrm{Var}\left[\frac{\chi^2_{n-2}}{n}\right] &= \frac{2(n-2)}{n^2} \to 0 && (2.33) \end{aligned}$$

with both limits happening as $n \to \infty$. In fact $\mathrm{Var}\left[\frac{\chi^2_{n-2}}{n}\right] = O(1/n)$, so

$$\hat{\sigma}^2 = \sigma^2\left(1 + O(1/\sqrt{n})\right) \quad (2.34)$$

Taking the square root, and using the fact that $(1+x)^a \approx 1 + ax$ when $|x| \ll 1$,

$$\hat{\sigma} = \sigma\left(1 + O(1/\sqrt{n})\right) \quad (2.35)$$

Put this together to look at our test statistic:

$$\begin{aligned} \frac{\hat{\beta}_1}{\hat{\mathrm{se}}\left(\hat{\beta}_1\right)} &= \frac{\beta_1 + O(1/\sqrt{n})}{\frac{\sigma\left(1+O(1/\sqrt{n})\right)}{s_X\sqrt{n}}} && (2.36)\\ &= \sqrt{n}\frac{\beta_1 + O(1/\sqrt{n})}{(\sigma/s_X)\left(1 + O(1/\sqrt{n})\right)} && (2.37)\\ &= \sqrt{n}\frac{\beta_1}{\sigma/s_X}\left(1 + O(1/\sqrt{n})\right) && (2.38)\\ &= \sqrt{n}\frac{\beta_1}{\sigma/s_X} + O(1) && (2.39) \end{aligned}$$

In words: so long as the true $\beta_1 \neq 0$, the test statistic is going to go off to $\pm\infty$, and the rate at which it escapes towards infinity is going to be proportional to $\sqrt{n}$. When we compare this against the null distribution, which is $N(0,1)$, eventually we'll get arbitrarily small p-values.

We can actually compute what those p-values should be, by two bounds on the standard Gaussian distribution:

$$\left(\frac{1}{x} - \frac{1}{x^3}\right)\frac{1}{\sqrt{2\pi}}e^{-x^2/2} < 1 - \Phi(x) < \frac{1}{x}\frac{1}{\sqrt{2\pi}}e^{-x^2/2} \tag{2.40}$$

Thus

$$\begin{aligned} P_n &= \mathbb{P}\left(|Z| \geq \left|\frac{\hat{\beta}_1}{\hat{\sigma}/\sqrt{n}s_X}a\right|\right) && (2.41)\\ &= 2\mathbb{P}\left(Z \geq \left|\frac{\hat{\beta}_1}{\hat{\sigma}/\sqrt{n}s_X}\right|\right) && (2.42)\\ &\leq \frac{2}{\sqrt{2\pi}}\frac{e^{-\frac{1}{2}\frac{\hat{\beta}_1^2}{\hat{\sigma}^2/ns_X^2}}}{\left|\frac{\hat{\beta}_1}{\hat{\sigma}/\sqrt{n}s_X}\right|} && (2.43)\end{aligned}$$

To clarify the behavior, let's take the logarithm and divide by n:

$$\begin{aligned} \frac{1}{n}\log P_n &\leq \frac{1}{n}\log\frac{2}{\sqrt{2\pi}} && (2.44)\\ &\quad -\frac{1}{n}\log\left|\frac{\hat{\beta}_1}{\hat{\sigma}/\sqrt{n}s_X}\right| \\ &\quad -\frac{1}{2n}\frac{\hat{\beta}_1^2}{\hat{\sigma}^2/ns_X^2} \\ &= \frac{\log\sqrt{2}\pi}{n} && (2.45)\\ &\quad +\frac{\log\left|\frac{\hat{\beta}_1}{\hat{\sigma}/s_x}\right|}{n} \\ &\quad -\frac{\log n}{2n} \\ &\quad -\frac{\hat{\beta}_1^2}{2\hat{\sigma}^2/s_X^2}\end{aligned}$$

Take the limit as $n \to \infty$:

$$\begin{aligned} \lim_{n\to\infty} \frac{1}{n} \log P_n &\leq \lim_n \frac{\log \sqrt{2}\pi}{n} \\ &\quad + \lim_n \frac{\log \frac{\hat{\beta}_1}{\hat{\sigma}/s_x}}{n} \\ &\quad - \lim_n \frac{\log n}{2n} \\ &\quad - \lim_n \frac{\hat{\beta}_1^2}{2\hat{\sigma}^2/s_X^2} \end{aligned} \tag{2.46}$$

Since $\hat{\beta}_1/(\hat{\sigma}/s_X) \to \beta_1/(\sigma/s_X)$, and $n^{-1}\log n \to 0$,

$$\lim_{n\to\infty} \frac{1}{n} \log P_n \leq -\frac{\beta_1^2}{2\sigma^2/s_X^2} \tag{2.47}$$

I've only used the upper bound on $1 - \Phi(x)$ from Eq. 2.40; if we use the lower bound from that equation, we get

$$\lim_{n\to\infty} \frac{1}{n} \log P_n \geq -\frac{\beta_1^2}{2\sigma^2/s_X^2} \tag{2.48}$$

Putting the upper and lower limits together,

$$\lim_{n\to\infty} \frac{1}{n} \log P_n = -\frac{\beta_1^2}{2\sigma^2/s_X^2}$$

Turn the limit around: at least for large n,

$$P_n \approx e^{-n\frac{\beta_1^2}{2\sigma^2/s_X^2}} \tag{2.49}$$

As we can see from the above display, the p-value depends on the magnitude of the coefficient ($|\beta_1|$), the sample size (n), the noise around the regression line (σ^2), and how spread out the data is along the x axis (s_X^2). Note that, *any* $\beta_1 \neq 0$ will (eventually) give exponentially small p-values. This means that with enough data, we can detect even arbitrarily small coefficients. However, it is not accurate to say that one coefficient is important because it has a really small p-value and another is unimportant because it's got a big p-value. We can only say that given the available data, this coefficient is statistically significant or not.

2.6 Confidence Sets and p-Values in R

When we estimate a model with `lm`, R makes it easy for us to extract the confidence intervals of the coefficients:

```
confint(object, level = 0.95)
```

Here `object` is the name of the fitted model object, and `level` is the confidence level; if you want 95% confidence, you can omit that argument. For instance:

```
library(gamair)
data(chicago)
death.temp.lm <- lm(death ~ tmpd, data = chicago)
confint(death.temp.lm)
##                   2.5 %     97.5 %
## (Intercept) 128.8783687 131.035734
## tmpd         -0.3096816  -0.269607
confint(death.temp.lm, level = 0.9)
##                     5 %        95 %
## (Intercept) 129.0518426 130.8622598
## tmpd         -0.3064592  -0.2728294
```

If you want p-values for the coefficients, those are conveniently computed as part of the `summary` function:

```
coefficients(summary(death.temp.lm))
##                Estimate Std. Error   t value     Pr(>|t|)
## (Intercept) 129.9570512 0.55022802 236.18763  0.00000e+00
## tmpd         -0.2896443 0.01022089 -28.33845 3.23449e-164
```

Notice how this actually gives us an array with four columns: the point estimate, the standard error, the t statistic, and finally the p-value. Each row corresponds to a different coefficient of the model. If we want, say, the p-value of the intercept, that's

```
coefficients(summary(death.temp.lm))[1, 4]
## [1] 0
```

The summary function will also print out a *lot* of information about the model:

```
summary(death.temp.lm)
##
## Call:
## lm(formula = death ~ tmpd, data = chicago)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -42.275  -9.018  -0.754   8.187 305.952
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept) 129.95705    0.55023  236.19   <2e-16 ***
```

```
## tmpd         -0.28964    0.01022  -28.34   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 14.22 on 5112 degrees of freedom
## Multiple R-squared:  0.1358,Adjusted R-squared:  0.1356
## F-statistic: 803.1 on 1 and 5112 DF,  p-value: < 2.2e-16
```

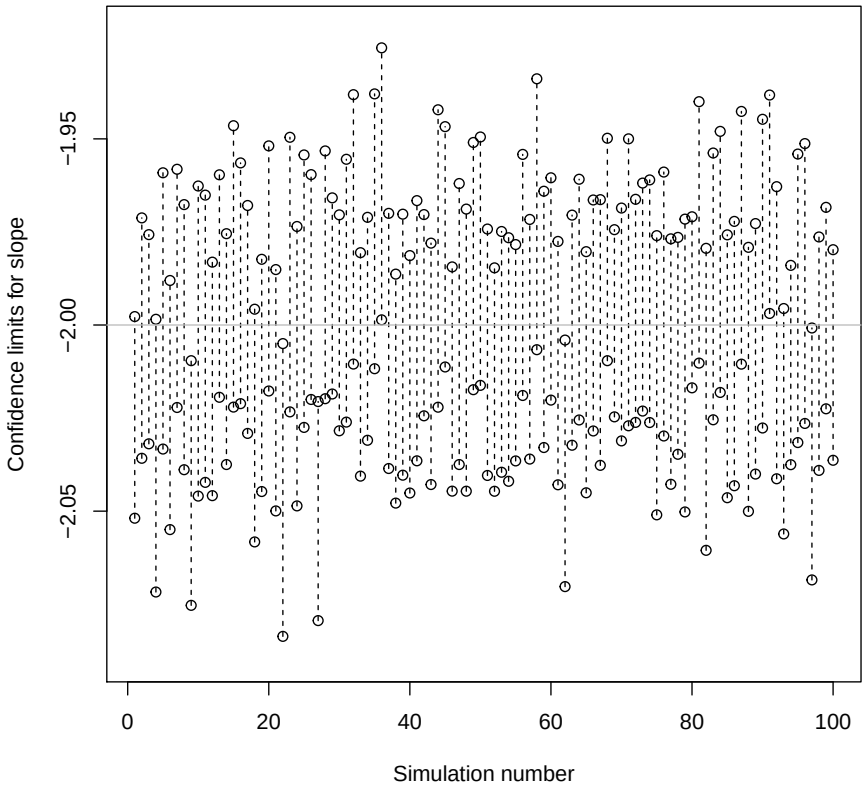
As my use of `coefficients(summary(death.temp.lm))` above suggests, the `summary` function actually returns a complex object, which can be stored for later access, and printed. Controlling how it gets printed is done through the `print` function:

```
print(summary(death.temp.lm), signif.stars = FALSE, digits = 3)
##
## Call:
## lm(formula = death ~ tmpd, data = chicago)
##
## Residuals:
##    Min     1Q Median     3Q    Max
## -42.27  -9.02  -0.75   8.19 305.95
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) 129.9571     0.5502   236.2   <2e-16
## tmpd         -0.2896     0.0102   -28.3   <2e-16
##
## Residual standard error: 14.2 on 5112 degrees of freedom
## Multiple R-squared:  0.136,Adjusted R-squared:  0.136
## F-statistic:  803 on 1 and 5112 DF,  p-value: <2e-16
```

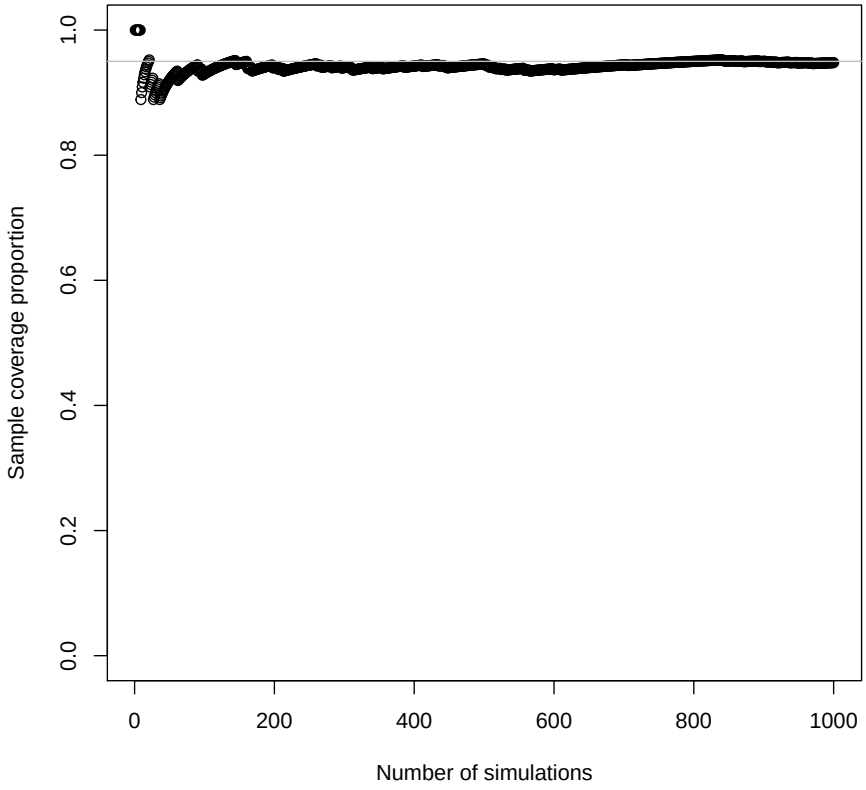
Coverage of the Confidence Intervals We carry out a computational demonstration of how the confidence interval for a parameter covers the true parameter value with the specified confidence level. We repeat many simulations of the model from Figure 2.2, calculate the confidence interval on each simulation, and plot those. We keep track of how often, in the first m simulations, the confidence interval covers the truth; this should converge to $1 - \alpha$ as m grows.

2.7 Predictive Inference for the Simple Linear Model

The downside of *point prediction* is that it does not give an idea of how far the truth is from our prediction. Confidence interval was introduced to tackle this point. The idea is to make an *interval* prediction with a coverage probability: "with such probability, Y will be in this interval".




```
CIs <- replicate(1000, confint(lm(y ~ x, data = sim.gnslrm(x = x, 5, -2, 0.1,
    FALSE)))[2, ])
plot(1:100, CIs[1, 1:100], ylim = c(min(CIs), max(CIs)), xlab = "Simulation number",
    ylab = "Confidence limits for slope")
points(1:100, CIs[2, 1:100])
segments(x0 = 1:100, x1 = 1:100, y0 = CIs[1, 1:100], y1 = CIs[2, 1:100], lty = "dashed")
abline(h = -2, col = "grey")
```




```
covered <- (CIs[1, ] <= -2) & (CIs[2, ] >= -2)
plot(1:length(covered), cumsum(covered)/(1:length(covered)), xlab = "Number of simulations",
    ylab = "Sample coverage proportion", ylim = c(0, 1))
abline(h = 0.95, col = "grey")
```

2.7.1 Confidence intervals for conditional means

The conditional mean at any particular x is just a number; we can do inference on it as though it were a parameter; it is, after all, a function of the parameters. More specifically, the true conditional mean is

$$m(x) \equiv \mathbb{E}[Y|X = x] = \beta_0 + \beta_1 x \tag{2.50}$$

while our estimate of the conditional mean is

$$\hat{m}(x) = \hat{\beta}_0 + \hat{\beta}_1 x \tag{2.51}$$

(See note on notation below.)

Recall that

$$\hat{m}(x) = \beta_0 + \beta_1 x + \frac{1}{n}\sum_{i=1}^{n}\left(1 + (x - \overline{x})\frac{x_i - \overline{x}}{s_X^2}\right)\epsilon_i \tag{2.52}$$

so that

$$\mathbb{E}[\hat{m}(x)] = \beta_0 + \beta_1 x = m(x) \tag{2.53}$$

and

$$\mathrm{Var}[\hat{m}(x)] = \frac{\sigma^2}{n}\left(1 + \frac{(x - \overline{x})^2}{s_X^2}\right) \tag{2.54}$$

Under the Gaussian noise assumption, $\hat{m}(x)$ is Gaussian (why?),

$$\hat{m}(x) \sim N\left(m(x), \frac{\sigma^2}{n}\left(1 + \frac{(x - \overline{x})^2}{s_X^2}\right)\right) \tag{2.55}$$

Notice how the variance grows as we move further and further away from the center of the data along the x axis. Also notice how all the unknown parameters show up on the right-hand side of the equation.

Exact confidence intervals At this point, getting confidence intervals for $m(x)$ works just like getting confidence intervals for β_0 or β_1: we use as our standard error

$$\hat{\mathrm{se}}(\hat{m}(x)) = \frac{\hat{\sigma}}{\sqrt{n-2}}\sqrt{1 + \frac{(x - \overline{x})^2}{s_X^2}} \tag{2.56}$$

and then find

$$\frac{\hat{m}(x) - m(x)}{\hat{\mathrm{se}}(\hat{m}(x))} \sim t_{n-2} \tag{2.57}$$

by entirely parallel arguments. $1 - \alpha$ confidence intervals follow as before as well.

2.7.2 Interpreting the confidence interval

This confidence interval has the same interpretation as one for the parameters: either

1. The true value of $m(x)$, i.e., the true value of $\mathbb{E}[Y|X=x]$, is in the interval, or
2. Something very unlikely happened when we got our data.

This is all well and good, but it does not tell us about how often future values of Y will be in this interval; it tells us about how often we capture the conditional average.

2.7.3 Large-n approximation

As n grows, the t distribution with $n-2$ degrees of freedom becomes, approximately, the standard Gaussian. It follows that for large n,

$$\frac{\hat{m}(x) - m(x)}{\widehat{\mathrm{se}}(\hat{m}(x))} \approx N(0,1) \tag{2.58}$$

so

$$\hat{m}(x) \approx N(m(x), \widehat{\mathrm{se}}(\hat{m}(x))^2) \tag{2.59}$$

and an approximate $1-\alpha$ confidence interval for $m(x)$ is

$$\hat{m}(x) \pm z_{\alpha/2}\frac{\hat{\sigma}}{\sqrt{n}}\sqrt{1+\frac{(x-\overline{x})^2}{s_X^2}} \tag{2.60}$$

(It doesn't matter whether we use $n-2$ or n in the denominator for $\widehat{\mathrm{se}}$.) Notice that the width of this interval $\to 0$ as $n \to \infty$.

2.7.4 Prediction Interval

A $1-\alpha$ **prediction interval** for $Y|X=x$ is a an interval $[l,u]$ where

$$\mathbb{P}(l \leq Y \leq u | X=x) = 1-\alpha \tag{2.61}$$

Since $Y|X=x \sim N(m(x),\sigma^2)$, it would be a simple matter to find these limits if we knew the parameters: the lower limit would be $m(x)+z_{\alpha/2}\sigma$, and the upper limit $m(x)+z_{1-\alpha/2}\sigma$. Unfortunately, we don't know the parameters.

However, we do know how the parameters are related to our estimates, so let's try to use that:

$$\begin{aligned}
Y|X=x &\sim N(m(x),\sigma^2) && (2.62)\\
&= m(x) + N(0,\sigma^2) && (2.63)\\
&= \hat{m}(x) + N\left(0, \frac{\sigma^2}{n}\left(1+\frac{(x-\overline{x})^2}{s_X^2}\right)\right) + N(0,\sigma^2) && (2.64)\\
&= \hat{m}(x) + N\left(0, \sigma^2\left(1+\frac{1}{n}+\frac{(x-\overline{x})^2}{n s_X^2}\right)\right) && (2.65)
\end{aligned}$$

```
sim.gnslrm <- function(x, intercept, slope, sigma.sq, mdl = TRUE) {
    n <- length(x)
    y <- intercept + slope * x + rnorm(n, mean = 0, sd = sqrt(sigma.sq))
    if (mdl) {
        return(lm(y ~ x))
    }
    else {
        return(data.frame(x = x, y = y))
    }
}
extract.pred.int <- function(mdl, x, level = 0.95) {
    predict(mdl, newdata = data.frame(x = x), interval = "prediction", level = level)
}
x.new <- 1/137
m <- 1000
alpha <- 0.05
beta.0 <- 5
beta.1 <- -2
sigma.sq <- 0.1
```

FIGURE 2.7: *Code setting up a simulation of a Gaussian-noise simple linear regression model, along a fixed vector of x_i values, followed by some default values we'll use in the later simulations.*

where in the last line I've used the fact that, under the assumptions of the model, the new Y we're trying to predict is independent of the old Y used to estimate the parameters. The variance, as we've seen, has two parts: the true noise variance about the regression line, plus the variance coming from our uncertainty in where that regression line is. Both parts of the variance are proportional to σ^2. Let's call the whole thing $\sigma^2_{pred}(x)$.

So, we have a random variable with a Gaussian distribution centered at $\hat{m}(x)$ and with a variance $\sigma^2_{pred}(x)$ proportional to σ^2. We can estimate that variance as

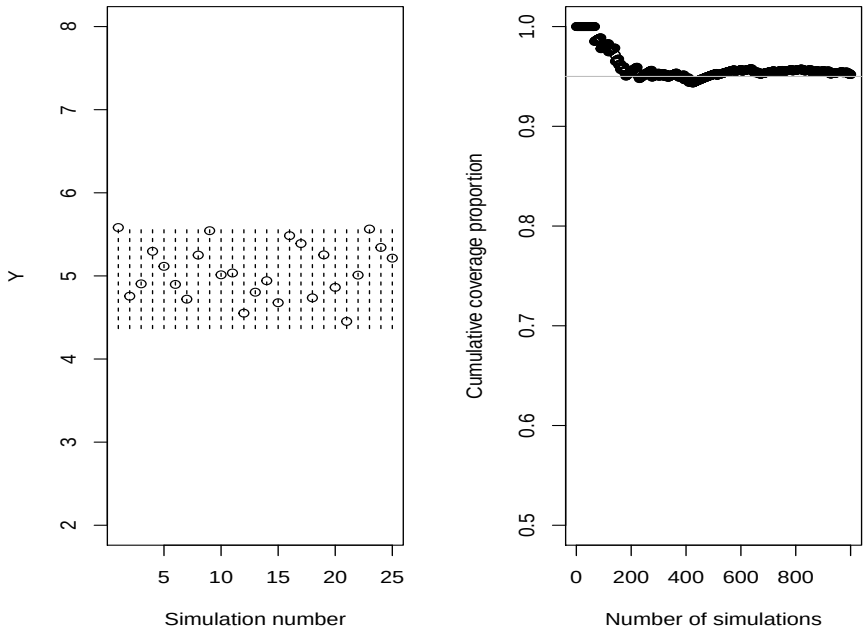
$$s^2_{pred}(x) = \hat{\sigma}^2 \frac{n}{n-2}\left(1 + \frac{1}{n} + \frac{(x-\overline{x})^2}{n s^2_X}\right) \tag{2.66}$$

Going through the now-familiar argument once again,

$$\frac{Y - \hat{m}(x)}{s_{pred}(x)} \mid X = x \sim t_{n-2} \tag{2.67}$$

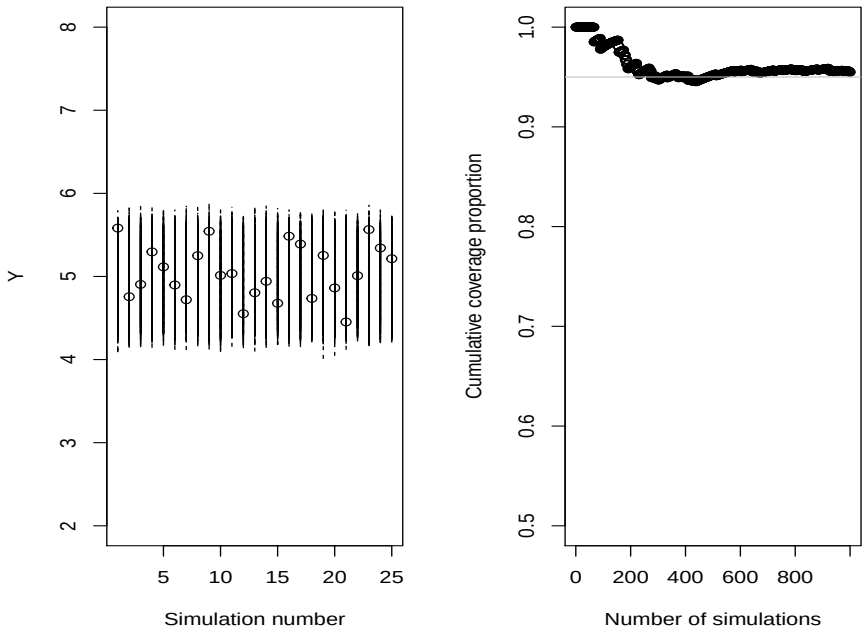
and we can use this to give prediction intervals.

Again, as usual, as $n \to \infty$, the t distribution turns into a standard Gaussian, while $s^2_{pred}(x) \to \sigma^2_{pred}(x) \to \sigma^2$. With *enough* data, then, our prediction intervals approach the ones we'd use if we knew the parameters and they were exactly our point estimates. Notice that the width of these prediction intervals does *not* go to zero as $n \to \infty$ — there is always some noise around the regression line!




```
y.new <- sim.gnslrm(x = rep(x.new, m), beta.0, beta.1, sigma.sq, mdl = FALSE)$y
pred.int <- beta.0 + beta.1 * x.new + sqrt(sigma.sq) * qnorm(c(alpha/2, 1 -
    alpha/2))
names(pred.int) <- c("lwr", "upr")
par(mfrow = c(1, 2))
plot(1:25, y.new[1:25], xlab = "Simulation number", ylab = "Y", ylim = c(2,
    8))
segments(x0 = 1:25, x1 = 1:25, y0 = pred.int["lwr"], y1 = pred.int["upr"], lty = "dashed")
covered <- (y.new >= pred.int["lwr"]) & (y.new <= pred.int["upr"])
plot(1:m, cumsum(covered)/(1:m), xlab = "Number of simulations", ylab = "Cumulative coverage proportion",
    ylim = c(0.5, 1))
abline(h = 1 - alpha, col = "grey")
par(mfrow = c(1, 1))
```

FIGURE 2.8: *Demonstration of the coverage of the prediction intervals. Here, we are seeing what would happen if we got to use the true coefficients, which are* $\beta_0 = 5$, $\beta_1 = -2$, $\sigma^2 = 0.1$*; we are always trying to predict* Y *when* $X = 1/137$.




```
x.seq <- seq(from = -5, to = 5, length.out = 42)
mdls <- replicate(m, sim.gnslrm(x = x.seq, beta.0, beta.1, sigma.sq, mdl = TRUE),
    simplify = FALSE)
pred.ints <- sapply(mdls, extract.pred.int, x = x.new)
rownames(pred.ints)[2:3] <- names(pred.int)
par(mfrow = c(1, 2))
plot(1:25, y.new[1:25], xlab = "Simulation number", ylab = "Y", ylim = c(2,
    8))
segments(x0 = 1:25, x1 = 1:25, y0 = pred.ints["lwr", ], y1 = pred.ints["upr",
    ], lty = "dashed")
covered <- (y.new >= pred.ints["lwr", ]) & (y.new <= pred.ints["upr", ])
plot(1:m, cumsum(covered)/(1:m), xlab = "Number of simulations", ylab = "Cumulative coverage pro
    ylim = c(0.5, 1))
abline(h = 1 - alpha, col = "grey")
par(mfrow = c(1, 1))
```

FIGURE 2.9: *As in Figure 2.8, but we are now using coefficients estimated by drawing 42 observations from the model, with the X's being evenly spaced from -5 to 5. Here, as you can see from the code, each prediction is made on the basis of a* different *random realization of the data before estimating the model. (See §2.8 below for details on how to use* `predict` *to return intervals.)*

2.7.5 Interpretation of the prediction interval

The interpretation of the prediction interval here is a bit tricky.

What we want for a prediction interval is that

$$\mathbb{P}(l \leq Y \leq u \mid X = x) = 1 - \alpha \tag{2.68}$$

Now our limits l and u involve the estimated parameters. To be explicit,

$$\mathbb{P}\left(\hat{m}(x) + t_{n-2}(\alpha/2)\hat{\text{se}}\left(\hat{m}(x)\right) \leq Y \leq \hat{m}(x) + t_{n-2}(1-\alpha/2)\hat{\text{se}}\left(\hat{m}(x)\right) \mid X = x\right) = 1-\alpha \tag{2.69}$$

But $\hat{m}(x)$ and $\hat{\text{se}}\left(\hat{m}(x)\right)$ are both random variables. The experiment we're imagining repeating when we write out Eq. 2.69 involves both estimating the parameters and predicting a new Y at $X = x$ every time.

If we estimate the parameters just once, and then try repeatedly measuring Y when $X = x$, we'll see that our coverage level, while close to $1 - \alpha$, is not quite $1 - \alpha$, sometimes less and sometimes more. The coverage gets closer to the desired level as the number of points n used to *estimate* the model grows.

2.8 Prediction intervals in R

For linear models, all of the calculations needed to find confidence intervals for $\hat{m}$ or prediction intervals for Y are automated into the `predict` function.

```
predict(object, newdata, interval = c("none", "confidence", "prediction"), level = 0.95)
```

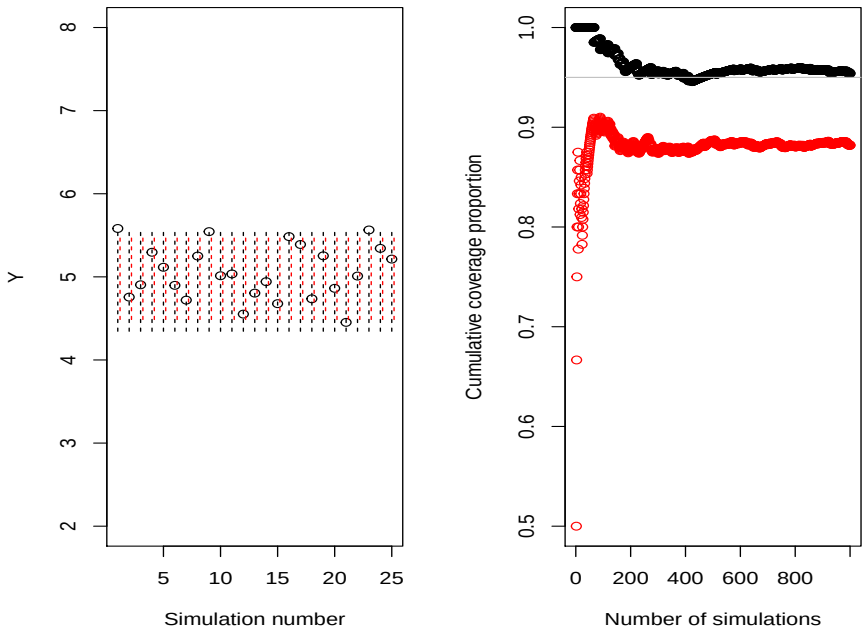
The `object` argument is the estimated model returned by `lm`; `newdata` is a data frame containing a column whose name matches that of the predictor variable. `interval` controls whether to give point predictions (`"none"`, the default) or intervals, and if so which kind. `level` is of course the confidence level (default 0.95 .)

To illustrate, consider the `chicago` data set:

```
library(gamair)
data(chicago)
death.temp.lm <- lm(death ~ tmpd, data = chicago)
```

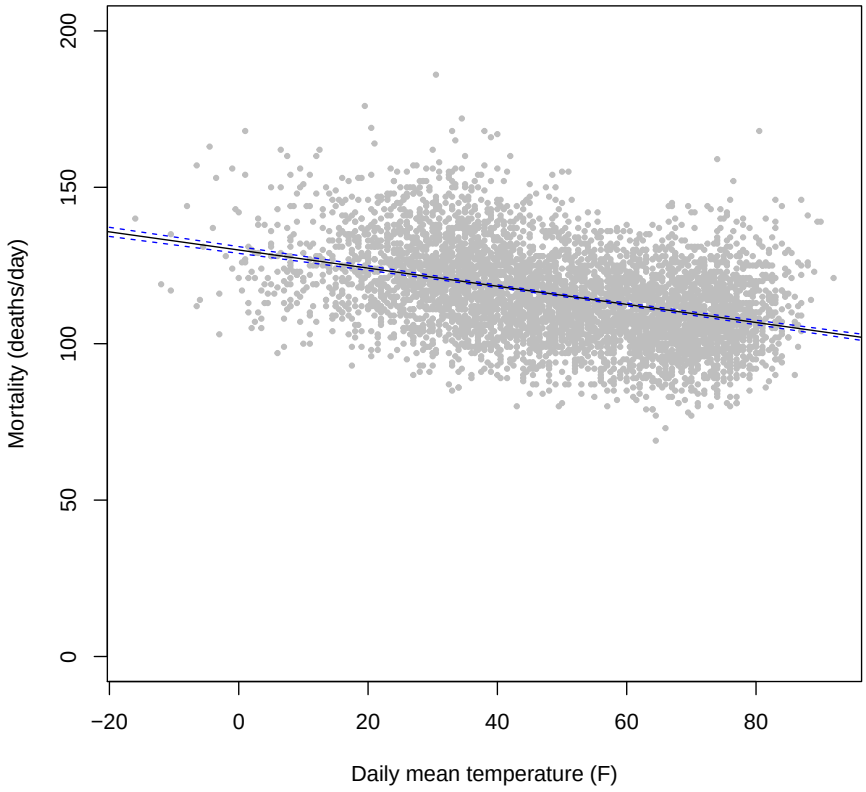
Figure 2.11 shows a scatter-plot of the data and the estimated line, together with confidence limits for the conditional mean at each point. Because we have thousands of data points and reasonably large s_X^2, the confidence limits are quite narrow, though you can see, from the plot, how they widen as we move away from the mean temperature.

Figure 2.12 shows the *prediction* limits for the same model. These are much wider, because their width is mostly coming from (the estimate of) σ, the noise around the regression line, the model being very confident that it knows what the line is. Despite their width, the bands don't include all the data points. This is not, in itself, alarming — they should only contain about 95% of the data points. I will leave it as an exercise to check what the actual coverage level is here.



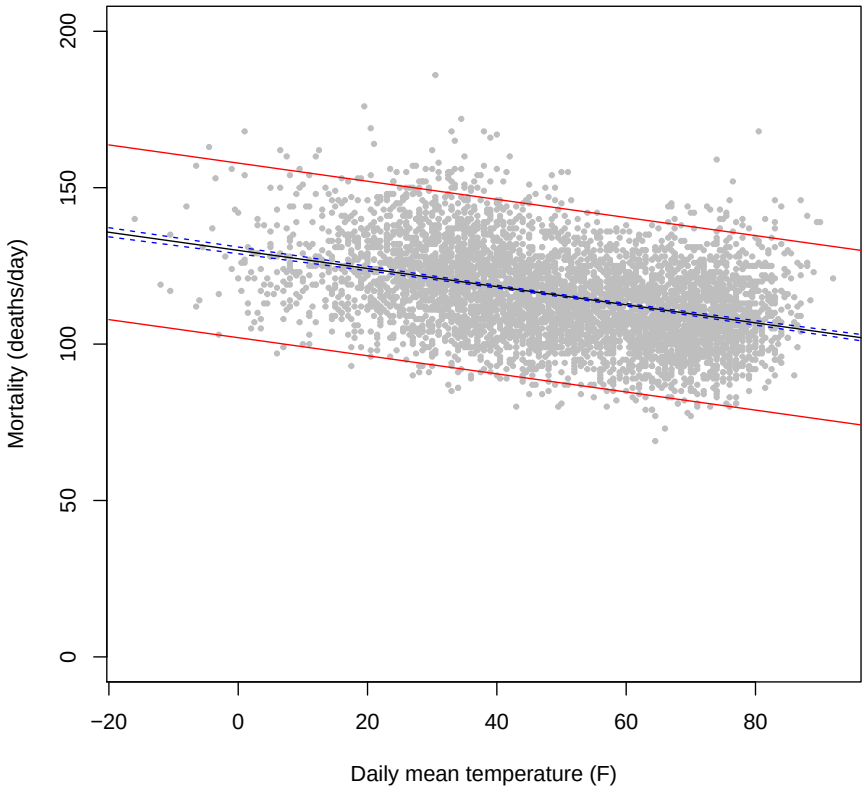

```
pred.ints <- sapply(mdls[1:2], extract.pred.int, x = x.new)
rownames(pred.ints)[2:3] <- c("lwr", "upr")
par(mfrow = c(1, 2))
plot(1:25, y.new[1:25], xlab = "Simulation number", ylab = "Y", ylim = c(2,
    8))
segments(x0 = 1:25, x1 = 1:25, y0 = pred.ints["lwr", 1], y1 = pred.ints["upr",
    1], lty = "dashed")
segments(x0 = 0.2 + 1:25, x1 = 0.2 + 1:25, y0 = pred.ints["lwr", 2], y1 = pred.ints["upr",
    2], lty = "dashed", col = "red")
covered.1 <- (y.new >= pred.ints["lwr", 1]) & (y.new <= pred.ints["upr", 1])
covered.2 <- (y.new >= pred.ints["lwr", 2]) & (y.new <= pred.ints["upr", 2])
plot(1:m, cumsum(covered.1)/(1:m), xlab = "Number of simulations", ylab = "Cumulative coverage p
    ylim = c(0.5, 1))
points(1:m, cumsum(covered.2)/(1:m), col = "red")
abline(h = 1 - alpha, col = "grey")
par(mfrow = c(1, 1))
```

FIGURE 2.10: *As in Figure 2.9, but all the new realizations of Y are being predicted based on the coefficients of one single estimate of the coefficients (the first estimate for the black intervals, the second estimate for the red). — The code for all three figures is very similar; could you write one function which, with appropriate arguments, would make all three of them?*




```
plot(death ~ tmpd, data = chicago, pch = 19, cex = 0.5, col = "grey", ylim = c(0,
    200), xlab = "Daily mean temperature (F)", ylab = "Mortality (deaths/day)")
abline(death.temp.lm)
temp.seq <- seq(from = -20, to = 100, length.out = 100)
death.temp.CIs <- predict(death.temp.lm, newdata = data.frame(tmpd = temp.seq),
    interval = "confidence")
lines(temp.seq, death.temp.CIs[, "lwr"], lty = "dashed", col = "blue")
lines(temp.seq, death.temp.CIs[, "upr"], lty = "dashed", col = "blue")
```

FIGURE 2.11: *Data from the Chicago death example (grey dots), together with the regression line (solid black) and the 95% confidence limits on the conditional mean (dashed blue curves). I have restricted the vertical range to help show the confidence limits, though this means some high-mortality days are off-screen.*




```
plot(death ~ tmpd, data = chicago, pch = 19, cex = 0.5, col = "grey", ylim = c(0,
    200), xlab = "Daily mean temperature (F)", ylab = "Mortality (deaths/day)")
abline(death.temp.lm)
temp.seq <- seq(from = -20, to = 100, length.out = 100)
death.temp.CIs <- predict(death.temp.lm, newdata = data.frame(tmpd = temp.seq),
    interval = "confidence")
lines(temp.seq, death.temp.CIs[, "lwr"], lty = "dashed", col = "blue")
lines(temp.seq, death.temp.CIs[, "upr"], lty = "dashed", col = "blue")
death.temp.PIs <- predict(death.temp.lm, newdata = data.frame(tmpd = temp.seq),
    interval = "prediction")
lines(temp.seq, death.temp.PIs[, "lwr"], col = "red")
lines(temp.seq, death.temp.PIs[, "upr"], col = "red")
```

FIGURE 2.12: *Adding 95% prediction intervals (red) to the previous plot.*

2.9 F-test

Definition 2. *The Fisher distribution with a, b degrees of freedom is defined to be the distribution of the ratio*

$$\frac{\chi_a^2/a}{\chi_b^2/b}$$

where χ_a^2 and χ_b^2 are independent.

The premise of this test is that the data set does follow the Gaussian linear model

$$Y = \beta_0 + \beta_1 X + \epsilon.$$

We want to ascertain wheter the slope parameter β_1 equals zero or not. In other words, we wonder whether we have a significant better fit of our data with the previous full model than with the simpler model

$$Y = \beta_0 + \epsilon$$

Formally, we are considering the following statistical testing problem:

$$\mathbf{H}_0 \,:\, \beta_1 = 0 \quad \text{vs} \quad \mathbf{H}_1 \,:\, \beta_1 \neq 0.$$

For the full model, we recall that the estimator of the variance is

$$\hat{\sigma}^2 = \frac{1}{n}\sum_{i=1}^{n}(y_i - (\hat{\beta}_0 + \hat{\beta}_1 x_i))^2$$

with

$$\frac{n\hat{\sigma}^2}{\sigma^2} \sim \chi^2_{n-2}.$$

For the null hypothesis model, the variance estimate is

$$\hat{\sigma}^2 = \frac{1}{n}\sum_{i=1}^{n}(y_i - \bar{y})^2 = \hat{\sigma}_Y^2,$$

and

$$\frac{n\hat{\sigma}_Y^2}{\sigma^2} \sim \chi^2_{n-1}.$$

It turns out that $\hat{s}_Y^2$ is not independent of $\hat{\sigma}^2$. However $\hat{s}_Y^2 - \hat{\sigma}^2$ is independent of $\hat{\sigma}^2$ and

$$\frac{n(\hat{s}_Y^2 - \hat{\sigma}^2)}{\sigma^2} \sim \chi^2_1.$$

Therefore, we have under the null hypothesis

$$\frac{\hat{s}_Y^2 - \hat{\sigma}^2}{\hat{\sigma}^2/(n-2)} \sim F_{1,n-1}.$$

Under the alternative hypothesis, this test statistic will follow a decentralized Fisher distributed with non-centrality parameter growing with $|\beta_1|$.

```
sim.gnslrm <- function(x, intercept, slope, sigma.sq, var.ratio = TRUE) {
    n <- length(x)
    y <- intercept + slope * x + rnorm(n, mean = 0, sd = sqrt(sigma.sq))
    if (var.ratio) {
        mdl <- lm(y ~ x)
        hat.sigma.sq <- mean(residuals(mdl)^2)
        return(s.sq.y/hat.sigma.sq)
    }
    else {
        return(data.frame(x = x, y = y))
    }
}
beta.0 <- 5
beta.1 <- 0
x.seq <- seq(from = -5, to = 5, length.out = 42)
```

FIGURE 2.13: *Code setting up a simulation of a Gaussian-noise simple linear regression model, returning either the actual simulated data frame, or just the variance ratio* $s_Y^2/\hat{\sigma}^2$.

Running this F test in R The easiest way to run the F test for the regression slope on a linear model in R is to invoke the anova function, like so:

```
anova(lm(y ~ x))
```

```
## Error in eval(expr, envir, enclos):  object 'y' not found
```

This will give us an analysis-of-variance table for the model. The actual object the function returns is an anova object, which is a special type of data frame. The columns record, respectively, degrees of freedom, sums of squares, mean squares, the actual F statistic, and the p value of the F statistic. What well care about will be the first row of this table, which will give us the test information for the slope on X. To illustrate more concretely, lets use again the Chicago data set:

```
library(gamair)
data(chicago)
death.temp.lm <- lm(death ~ tmpd, data = chicago)
anova(death.temp.lm)
```

```
f.stats <- replicate(1000, anova(lm(y ~ x, data = sim.gnslrm(x.seq, beta.0,
    beta.1, sigma.sq, FALSE)))[1, 4])
null.hist <- hist(f.stats, breaks = 50, plot = FALSE)
alt.f <- replicate(1000, anova(lm(y ~ x, data = sim.gnslrm(x.seq, beta.0, -0.05,
    sigma.sq, FALSE)))[1, 4])
alt.hist <- hist(alt.f, breaks = 50, plot = FALSE)
plot(0, xlim = c(0, 30), ylim = c(0, 1.2), xlab = "F", ylab = "Density", type = "n")
plot(alt.hist, freq = FALSE, add = TRUE, col = "grey", border = "grey")
plot(null.hist, freq = FALSE, add = TRUE)
curve(df(x, 1, length(x.seq) - 2), add = TRUE, col = "blue")
```

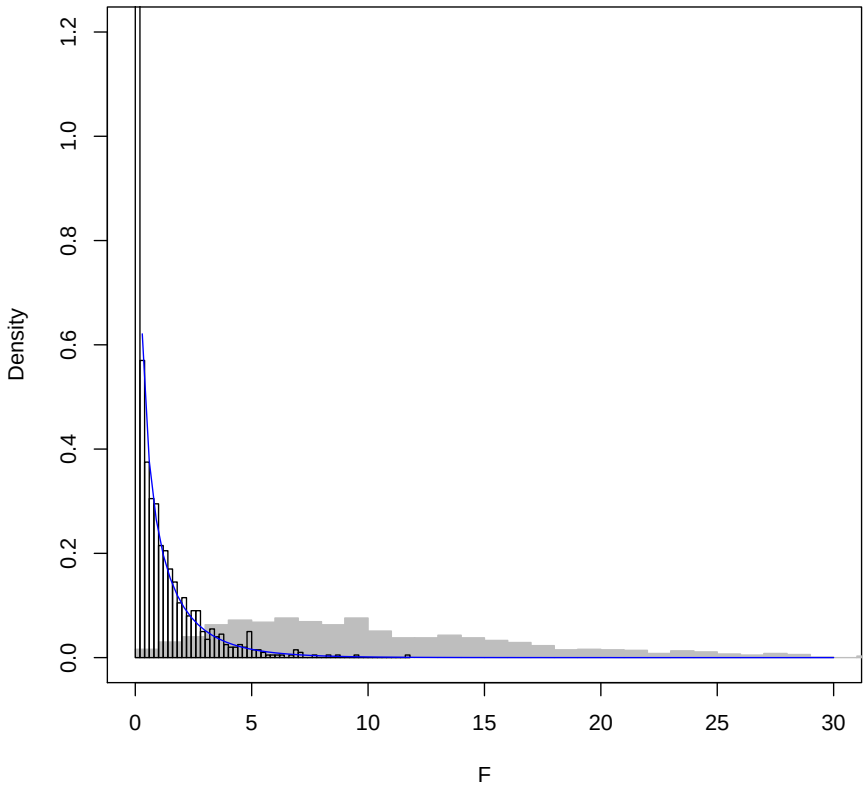


FIGURE 2.14: *Comparing the actual distribution of F statistics when we simulate under the null model (black histogram) to the theoretical $F_{1,n2}$ distribution (blue curve), and to the distribution under the alternative $\beta_1 = 0.05$.*

```
p.vals <- pf(f.stats, 1, length(x.seq) - 2, lower.tail = FALSE)
alt.p <- pf(alt.f, 1, length(x.seq) - 2, lower.tail = FALSE)
hist(alt.p, col = "grey", freq = FALSE, xlab = "p-value", main = "", border = "grey",
    xlim = c(0, 1))
plot(hist(p.vals, plot = FALSE), add = TRUE, freq = FALSE)
```

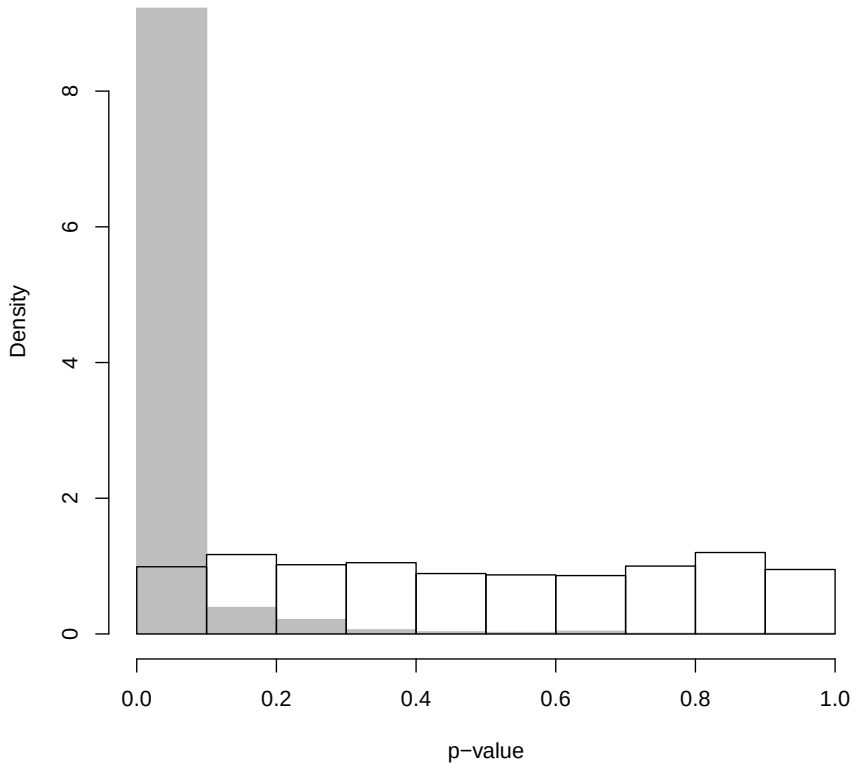


FIGURE 2.15: *Distribution of p-values from repeated simulations, under the null hypothesis (black) and the alternative (grey). Notice how the p-values under the null are uniformly distributed, while under the alternative they are bunched up towards small values at the left.*

Chapter 3

Diagnostics and Modifications for Simple Regression

In the previous chapters, we have laid out what regression analysis is for, why we use statistical models to do it, the assumptions of the simple linear regression model, and estimation and prediction for the simple regression model using both the method of maximum likelihood and the method of least squares. We could, at this point, follow the traditional route in a class like this of plunging into detailed inferential calculations for the model (this is how you find a confidence interval for such-and-such, etc.). However, those calculations have little point if the assumptions for the model do not hold. Thus, we will look at model checking *first*, and in later chapters go on to the inferential theory.

It is somewhat more traditional to talk about "diagnostics" for the regression model, rather than "model checking"; which name is best depends on how formal you want to sound. Just as in medicine, a good diagnostic procedure is one which will *differentiate* between health (for us: the model assumptions hold) and illness (the assumptions are violated). Indeed, a really good diagnostic procedure will distinguish between different sorts of illness (tell us *which* assumptions are violated). More mathematically, we want to find consequences of the assumptions which will hold no matter what the parameter values might be, but would be hard to achieve if the assumptions fail. (Checking whether $\sum_i \hat{\beta}_0 + \hat{\beta}_1 x_i = \sum_i y_i$ is not a useful diagnostic, because while that will be true if the assumptions are all right, it would still be true if they were all wrong.) The simplest and most useful check all involve the residuals.

3.1 The Residuals

In previous chapters, we defined the **residual** at the i^{th} data point as the difference between the realized value of the response y_i and what the estimated model would predict:

$$e_i = y_i - \hat{m}(x_i)$$

which, for the simple linear regression model, is just

$$e_i = y_i - (\hat{\beta}_0 + \hat{\beta}_1 x_i)$$

Residuals vs. the noise The residuals are very closely related to the noise variables ϵ_i, but with some important differences. If we take the basic equation for the simple linear model, $Y_i = \beta_0 + \beta_1 x_i + \epsilon_i$, we can re-arrange it to read

$$\epsilon_i = Y_i - (\beta_0 + \beta_1 x_i)$$

This has the same form as the equation for the residuals, but it involves the true parameters, not their estimates. If we take that equation for the i^{th} residual and substitute in the equation for Y_i,

$$e_i = \beta_0 + \beta_1 x_i - (\hat{\beta}_0 + \hat{\beta}_1 x_i) + \epsilon_i \tag{3.1}$$

$$(\beta_0 - \hat{\beta}_0) + (\beta_1 - \hat{\beta}_1) x_i + \epsilon_i \tag{3.2}$$

The terms in parentheses on the right-hand side are hopefully small, but they're not (in general) zero.

Residuals as weighted sums of noise To understand what's going on with the residuals, it's helpful to write them as a weighted sum of the ϵ's. Going back to previous chapters,

$$\hat{\beta}_0 = \beta_0 + \frac{1}{n}\sum_{i=1}^{n}\left(1 - \overline{x}\frac{x_i - \overline{x}}{s_X^2}\right)\epsilon_i \tag{3.3}$$

$$\hat{\beta}_1 = \beta_1 + \sum_{i=1}^{n}\frac{x_i - \overline{x}}{n s_X^2}\epsilon_i \tag{3.4}$$

Substitute these in to the equation for e_i:

$$e_i = \epsilon_i + \frac{1}{n}\sum_{j=1}^{n}\left(1 - \overline{x}\frac{x_j - \overline{x}}{s_X^2}\right)\epsilon_j + x_i \sum_{j=1}^{n}\frac{x_j - \overline{x}}{n s_X^2}\epsilon_j \tag{3.5}$$

$$= \sum_{j=1}^{n}\left(\delta_{ij} + \frac{1}{n} + (x_i - \overline{x})\frac{x_j - \overline{x}}{n s_X^2}\right)\epsilon_j \tag{3.6}$$

using the "Kronecker delta" ($\delta_{ij} = 0$ if $i \neq j$, $= 1$ if $i = j$) and some algebra. The factor in parenthesis is a weight which gets applied to each ϵ_j when summing

them up — a weight which depends on the x's alone. Let's abbreviate this weight by c_{ij}.

One of the assumptions of the simple linear model is that $\mathbb{E}[\epsilon_i|X] = 0$. Since we've shown that e_i is a weighted sum of ϵ_j, it follows that

$$\mathbb{E}[e_i|X] = \sum_j c_{ij}\mathbb{E}[\epsilon_j|X] = 0$$

Since the simple linear model assumes that the ϵ's are uncorrelated and all have variance σ^2, even conditional on X,

$$\begin{aligned} \mathrm{Var}[e_i|X] &= \sum_j \mathrm{Var}[c_{ij}\epsilon_j|X] && (3.7)\\ &= \sum_j c_{ij}^2 \mathrm{Var}[\epsilon_j|X] && (3.8)\\ &= \sigma^2 \sum_{j=1}^{n} c_{ij}^2 && (3.9) \end{aligned}$$

(I will not bother writing out the sum explicitly.) From here, one can go on to show

$$\mathrm{Var}[e_i] = \frac{n-2}{n}\sigma^2$$

though again I omit the details, so as not to spoil a future assignment.

If we make the Gaussian noise assumption, the ϵ_j are independent Gaussians. It thus follows that e_i also has a Gaussian distribution.

Contrast between residuals and noise terms The sum of the noise terms which produced the data is rarely zero. The *expectation value* of the sum of the noise is zero,

$$\mathbb{E}\left[\sum_{i=1}^{n}\epsilon_i\right] = \sum_{i=1}^{n}\mathbb{E}[\epsilon_i] = 0$$

but the *variance* is not:

$$\mathrm{Var}\left[\sum_{i=1}^{n}\epsilon_i\right] = \sum_{i=1}^{n}\mathrm{Var}[\epsilon_i] = n\sigma^2$$

so the sum of the noise terms can't be exactly zero all the time:

$$\sum_{i=1}^{n}\epsilon_i \neq 0$$

(Indeed, if the ϵ_i follow a Gaussian distribution, then $\sum_{i=1}^{n}\epsilon_i \sim N(0, n\sigma^2)$, and the probability that $\sum_{i=1}^{n}\epsilon_i = 0$ is zero, not one.) Similarly, while the ϵ are uncorrelated with X,

$$\mathrm{Cov}[X, \epsilon] = \mathbb{E}[X\epsilon] - \mathbb{E}[\epsilon]\,\mathbb{E}[X] = 0$$

the ϵ_i don't have a zero *sample* correlation with X:

$$\sum_{i=1}^{n} \epsilon_i (x_i - \overline{x}) \neq 0$$

On the other hand, such equations do hold exactly and deterministically for the residuals. In particular,

$$\sum_{i=1}^{n} e_i = 0$$

when the simple linear model is estimated by least squares, no matter what. This is because this equation is a consequence of the estimating equations and the estimating equations alone — it applies all the time, on any data set, not just with probability 1 but without exception. Similarly,

$$\sum_{i=1}^{n} (x_i - \overline{x}) e_i = 0$$

also follows from the estimating equations. I will not go over the derivations, so as not to spoil the homework you are currently doing.

(If we think of $(e_1, e_2, \ldots e_n)$ as an n-dimensional vector, these two equations tell us that not every vector is possible. Only vectors which live in an $(n-2)$-dimensional subspace are allowed. This is, so to speak, the same $n-2$ as in $\mathrm{Var}[e_i]$.)

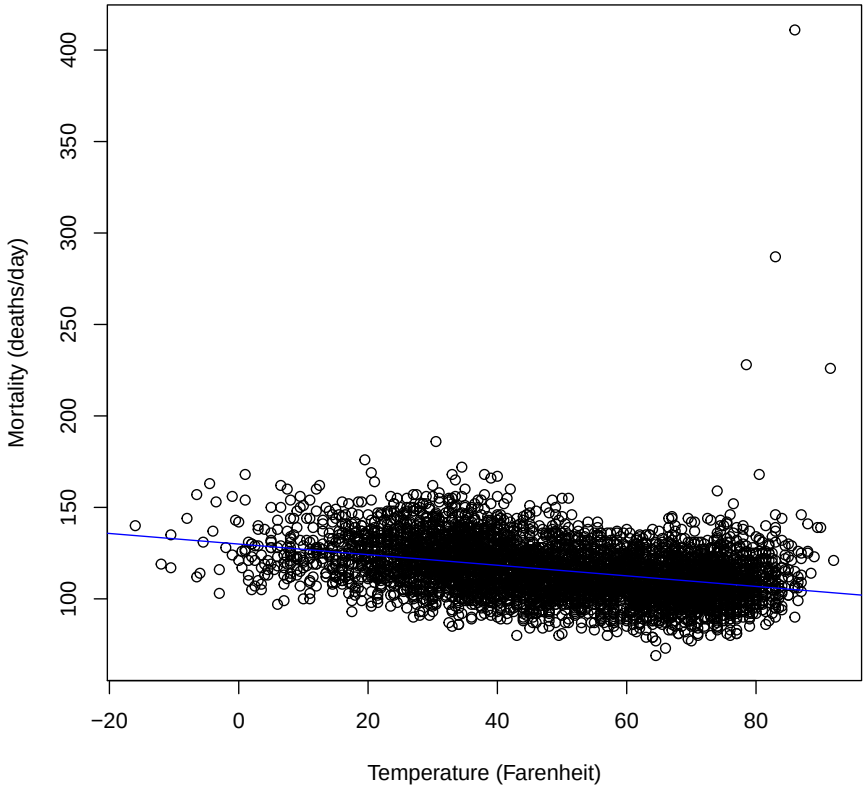
These two equations imply that even when the ϵ's are independent, the residuals are not. (After all, if we know all but one residual, the last one is completely determined.) However, the dependence is typically very slight and subtle, and it gets weaker as n grows (because each e_i is making a comparatively-small contribution to the sum that must be zero). So the residuals should show only negligible correlation.

3.1.1 Summary on Properties of the Residuals

Let's sum up the most relevant observations from the last couple of paragraphs.

1. The residuals should have expectation zero, conditional on x, $\mathbb{E}[e_i | X = x] = 0$. (The residuals should also have an over-all sample mean of exactly zero.)
2. The residuals should show a constant variance, unchanging with x.
3. The residuals can't be completely uncorrelated with each other, but the correlation should be extremely weak, and grow negligible as $n \to \infty$.
4. If the noise is Gaussian, the residuals should also be Gaussian.

Each one of these points leads to a diagnostic, to a check on the model. These take the form of our plots, which you should always, always make for any regression you run.

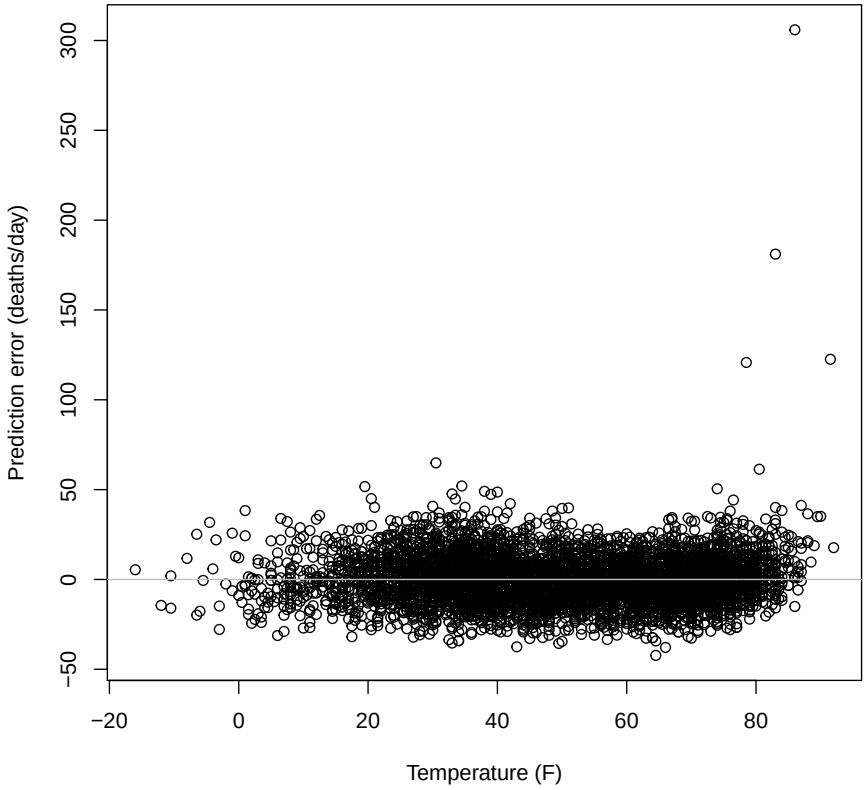



```
library(gamair)
data(chicago)
plot(death ~ tmpd, data = chicago, xlab = "Temperature (Farenheit)", ylab = "Mortality (deaths/day)")
death.temp.lm <- lm(death ~ tmpd, data = chicago)
abline(death.temp.lm, col = "blue")
```

FIGURE 3.1: *Plot of the data from the last homework, used as a running example, along with the estimated linear model (in blue).*

3.1.2 Plot the Residuals Against the Predictor

Make a scatter-plot with the residuals on the vertical axis and the predictor variable on the horizontal axis. Because $\mathbb{E}[e|X = x] = 0$, and $\mathrm{Var}[e|X = x]$ is constant, this should, ideally, look like a constant-width blur of points around a straight, flat line at height zero. Deviations from this — changing width, curvature, substantial regions of the x axis where the average residuals are either positive or negative — are all signs that the model is mis-specified. In particular, curved or stepped patterns indicate that $\mathbb{E}[e|X = x] \neq 0$, which in turn means that $\mathbb{E}[\epsilon|X = x] \neq 0$, which means that the simple-linear part of the simple linear regression model is wrong. One needs either more predictor variables (getting us into multiple regression), or a different functional form for the regression (getting us into nonlinear regression), or both.




```
plot(chicago$tmpd, residuals(death.temp.lm), xlab = "Temperature (F)", ylab = "Prediction error (deaths/day)
abline(h = 0, col = "grey")
```

FIGURE 3.2: *Residuals (vertical axis) vs. the predictor variable of temperature (horizontal axis).*

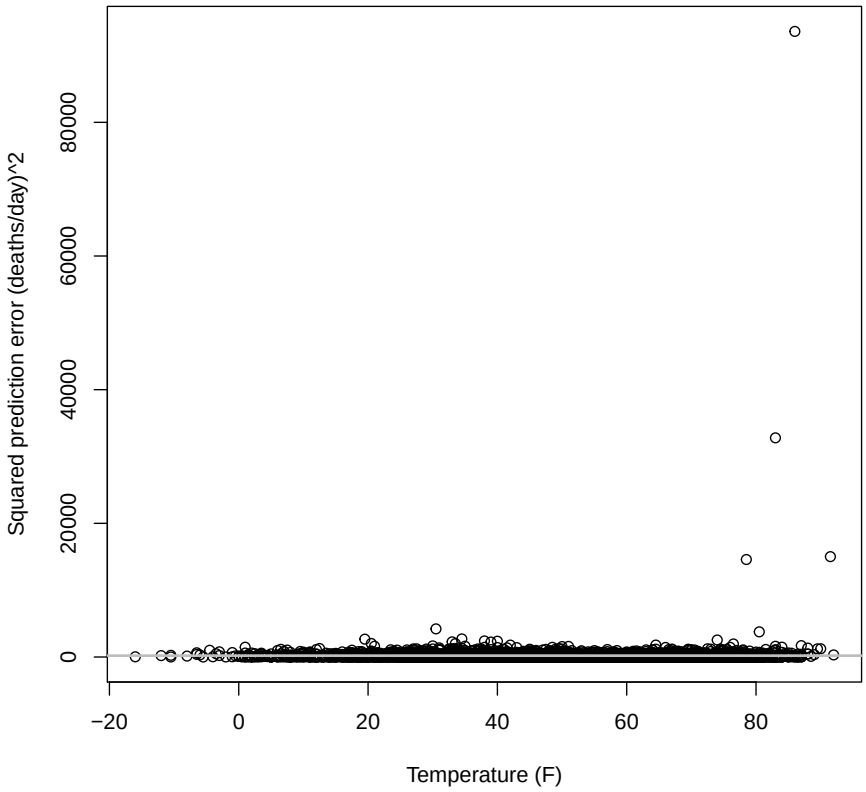
Plotting Against Another Variable If you have other potential predictor variables, you should be able to plot the residual against them, and also see a flat line around zero. If not, that's an indication that the other variable does in fact help predict the response, and so you should probably incorporate that in your model. (Part of the residuals from your simple linear model are really the contributions of that other predictor, which you were treating as noise out of ignorance.) In particular, if you make such a plot and you see the points in it fall around a straight line, that's an excellent sign that you need a multiple linear regression model.

(Exercise: make a plot of the residuals from this model against one of the pollution variables from the data set. Does it look like noise?)

3.1.3 Plot the Magnitude of the Residuals Against the Predictor

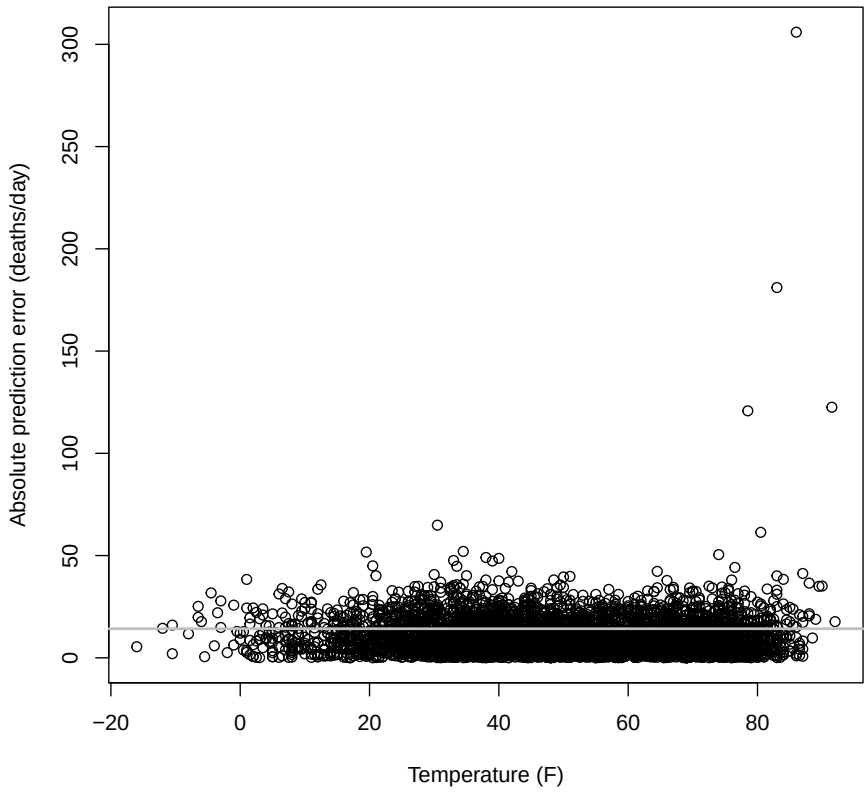
Because $\mathbb{E}[e|X = x] = 0$, $\mathrm{Var}[e|X = x] = \mathbb{E}\left[e^2|X = x\right]$. (Why?) This means we can check whether the variance of the residuals is constant by plotting the *squared* residuals against the predictor variable. This should give a scatter of points around a flat line, whose height should be around the in-sample MSE. Regions of the x axis where the residuals are persistently above or below this level are evidence of a problem with the simple linear regression model. This could be due to non-constant noise variance ("heteroskedasticity", in the jargon), or due to getting the functional form of the regression wrong. One can often get a clue as to what is driving the problem by looking to see whether the regions where the squared residuals are too big are also regions where the residuals are persistently above or below zero.

Sometimes, particularly when the model is not doing so well, squaring the residuals leads to a visually uninformative plot, because big residuals lead to really, really big squared residuals, and it's hard to make out any detail. A common fix is to then plot the absolute value of the residuals, with the reference horizontal line being at the square root of the mean squared error.




```
plot(chicago$tmpd, residuals(death.temp.lm)^2, xlab = "Temperature (F)", ylab = "Squared prediction error (
abline(h = mean(residuals(death.temp.lm)^2), lwd = 2, col = "grey")
```

FIGURE 3.3: *Squared residuals vs. temperature.*




```
plot(chicago$tmpd, abs(residuals(death.temp.lm)), xlab = "Temperature (F)",
    ylab = "Absolute prediction error (deaths/day)")
abline(h = sqrt(mean(residuals(death.temp.lm)^2)), lwd = 2, col = "grey")
```

FIGURE 3.4: *Absolute residuals vs. temperature; plotting these rather than the squared residuals reduces the* visual *impact of the few huge residuals.*

3.1.4 Plot the Residuals Against Coordinates and Each Other

Lots of the time, our data were collected in a certain order — each data point has some coordinates, in space or in time or both. Under the simple linear regression model, these shouldn't matter[1], so you should always plot the residuals against the coordinates. (Even if you have no coordinates, you should always plot residuals against the row numbers of the data set.) Clusters of nearby observations with unusually high or low residuals are a bad sign.

Of course, a certain amount of apparent clustering will happen naturally in any random process, so if this looks worrisome, one should really do some sort of formal test. Fortunately, there are lots of good test procedures for finding "runs" in what should be random noise. A quick hack, though, is simply to put the residuals in a totally random order and re-plot them:

```
sample(residuals(my.model))
```

will take the residuals vector of `my.model` and randomly permute it; plotting these permuted residuals against the coordinate will then give an example of how things should look. Do this a few times, and you'll get a good sense of how much apparent clumping of residuals should be produced by chance. This trick can also be used when plotting the residuals, or squared residuals, against the predictor variable, and can be formalized as what's called a "permutation test".

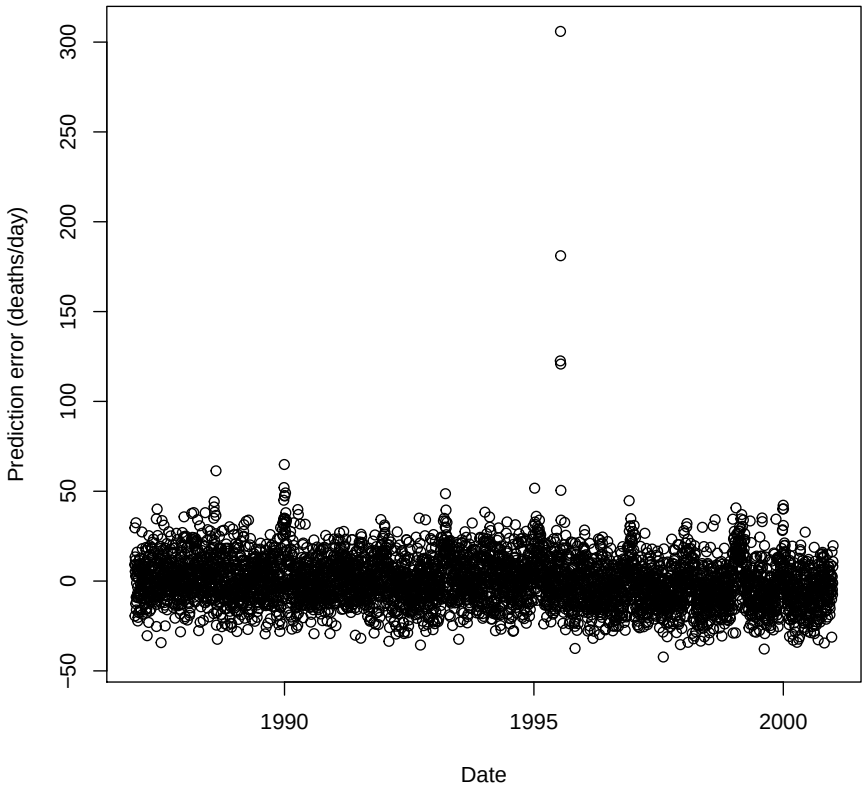
(Exercise: Make a few plots of the permuted, shuffled residuals against date.)

Residuals vs. Residuals A related diagnostic plot is particularly useful when the observations are taken at regular intervals along some axis (usually time but occasionally distance): make a scatter-plot with one point for each observation except the very last; the horizontal coordinate comes from that point's residual, and the vertical coordinate comes from the *next* point's residual. Ideally, this should give a blob of points with no particular structure. If the residuals are Gaussian, follow any other bell-ish distribution, it should show a circular blob. (If they were uniform, the blob should fill out a square — why?) Falling along a line or curve, or even a tilted ellipse, would be an indication of correlation between successive residuals, which in turn may be a sign that the noise is correlated[2].

Again, if you're not sure whether you're looking at a worryingly big departure from blob-hood, try permuting the residuals before re-plotting.

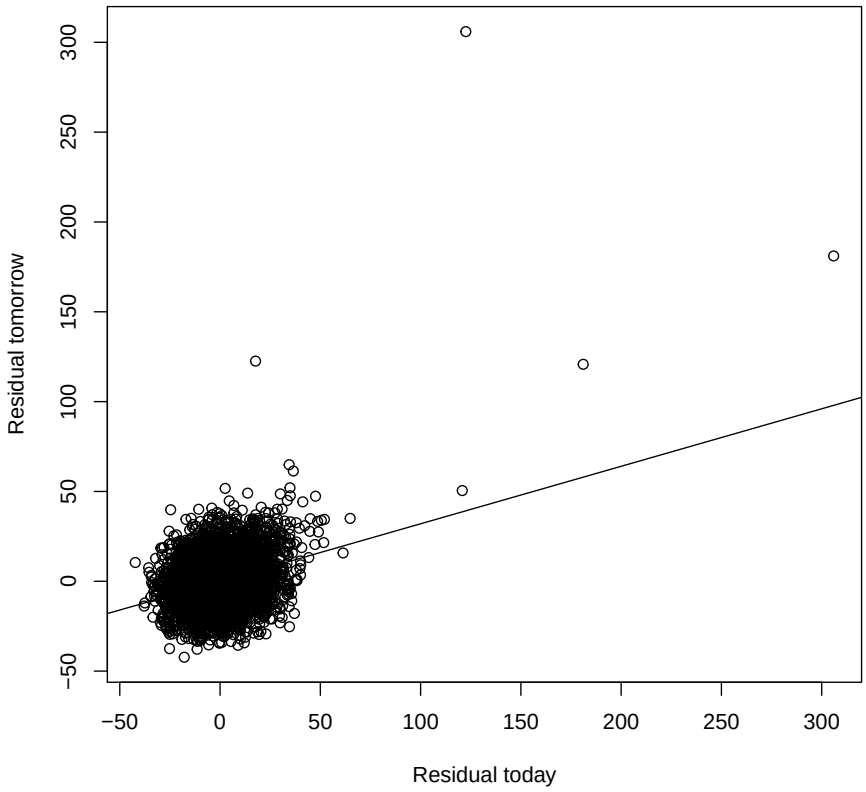
[1] Unless the predictor variable is one of the coordinates, of course.

[2] Or, again, it could be a sign that we've got the functional form wrong, so it's our systematic error which is correlated — see §3.5.




```
plot(as.Date(chicago$time, origin = "1993-12-31"), residuals(death.temp.lm),
    xlab = "Date", ylab = "Prediction error (deaths/day)")
```

FIGURE 3.5: *Residuals vs. date.*




```
plot(head(residuals(death.temp.lm), -1), tail(residuals(death.temp.lm), -1),
    xlab = "Residual today", ylab = "Residual tomorrow")
abline(lm(tail(residuals(death.temp.lm), -1) ~ head(residuals(death.temp.lm),
    -1)))
```

FIGURE 3.6: *Residuals for each day (except the first) plotted as a function of the residuals of the day before. The straight line shows a regression of tomorrow's residual on today's residual, which ideally should be a totally flat line.*

3.1.5 Plot the Distribution of the Residuals

Under the Gaussian noise assumption, the residuals should also follow a Gaussian distribution. We should therefore make plots of the distribution of the residuals, and compare that to a Gaussian.

The most basic plot of the distribution for the residuals is of course a histogram. This should be over-laid with a Gaussian probability density — but which Gaussian? The most reasonable one has mean 0 (because we know the residuals average to 0), and the same standard deviation as the residuals (because that's the MLE of the standard deviation in a Gaussian model). At that point, one can *see* whether the distribution of residuals looks like that of the best-fitting Gaussian.

Q-Q plots An alternative is what's called a "quantile-quantile" or "Q-Q" plot. (The textbook, old-fashionedly, calls this a "normal probability plot.) This takes a bit more thought to get used to than visually comparing density estimates, but with practice becomes most sensitive and less subjective. Here's the idea.

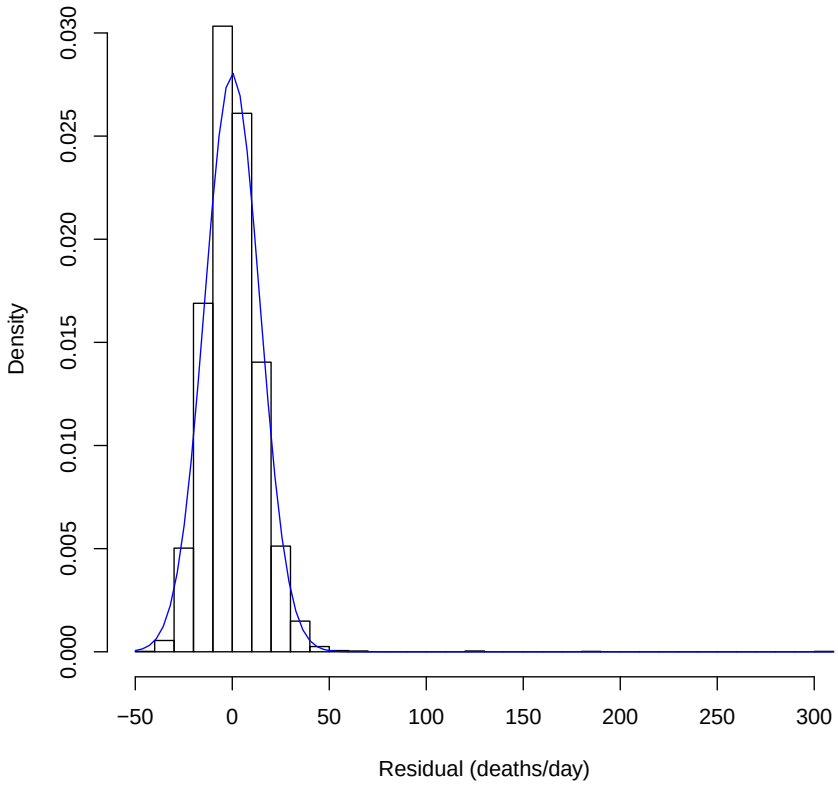
As you remember from intro. prob., knowing the cumulative distribution function (CDF) tells us all there is to know, mathematically, about a probability distribution; call this $F(x) = \mathbb{P}(X \leq x)$. If the distribution is continuous, the CDF has an inverse function, F^{-1}, where $F^{-1}(p)$ is the unique x such that $\mathbb{P}(X \leq x) = p$. This is called the **quantile function** — it tells us what level of x will contain a fraction p of the total probability. Since saying things like "the 0.95 quantile" is rather awkward, we usually pronounce it as "the 95^{rm} percentile", meaning the value greater than or equal to 95% of the population. If we know the quantile function, we can invert it to get the CDF, so the quantile function also completely determines the probability distribution[3].

As p varies from 0 to 1, $F^{-1}(p)$ will vary from the smallest possible value for the distribution up to the largest possible value. If our distribution is a Gaussian, with mean μ and variance σ^2, then $F^{-1}(p) = \sigma\Phi^{-1}(p) + \mu$, where Φ is the standard Gaussian CDF. (Why?) So if instead of plotting F^{-1} against p, we make a plot where $F^{-1}(p)$ goes on one axis and $\Phi^{-1}(p)$ goes on the other, as we sweep p from 0 to 1 we'll get a straight line. Conversely, if we weren't sure whether the distribution F we were interested in was Gaussian, but we did one of these quantile-quantile plots against the standard Gaussian and a got a straight line, then we'd know F was, in fact, Gaussian.

With a finite sample from a distribution (like, say, the vector of residuals), we don't really have F or F^{-1}. However, we can use the **sample quantiles** or **empirical quantiles**. Start with our observations, say $x_1, x_2, \ldots x_n$. Now put them in increasing order: to help distinguish the ordered from unordered observations, I'll use a common convention where the subscripts in parentheses show order, so

$$x_{(1)} \leq x_{(2)} \leq \ldots \leq x_{(n-1)} \leq x_{(n)}$$

[3] In R, the CDF and quantile functions have names beginning with `p` and `q` — `pnorm` and `qnorm` for the Gaussian, `pexp` and `qexp` for the exponential, etc.




```
hist(residuals(death.temp.lm), breaks = 40, freq = FALSE, xlab = "Residual (deaths/day)",
    main = "")
curve(dnorm(x, mean = 0, sd = sd(residuals(death.temp.lm))), add = TRUE, col = "blue")
```

FIGURE 3.7: *Histogram of the residuals, on a density scale, and the theoretical Gaussian distribution with the same mean (0) and standard deviation.*

(These ordered values of the data are sometimes called the **order statistics**.) Now $x_{(i)}$ is $\geq$ a fraction i/n of the sample observations, so $\hat{F}^{-1}(i/n) = x_{(i)}$. When we make a $Q-Q$ plot against a Gaussian distribution, we therefore put $x_{(i)}$ on one axis, and $\Phi^{-1}(i/n)$ on the other, and hope to see a straight line if the distribution of x is indeed Gaussian[4]

To sum up: we put the data in order, and then plot $x_{(i)}$ against $\Phi^{-1}(i/n)$. If the data are from a Gaussian distribution, these points should fall along a straight line. Small wiggles around the line are to be anticipated from chance; systematic deviations from a line indicate systematic departures from a Gaussian distribution.

Q-Q plots for other distributions One can make a Q-Q plot for data against any other distribution one likes, provided one knows the reference distribution's quantile function; R just provides code for the Gaussian case, however.

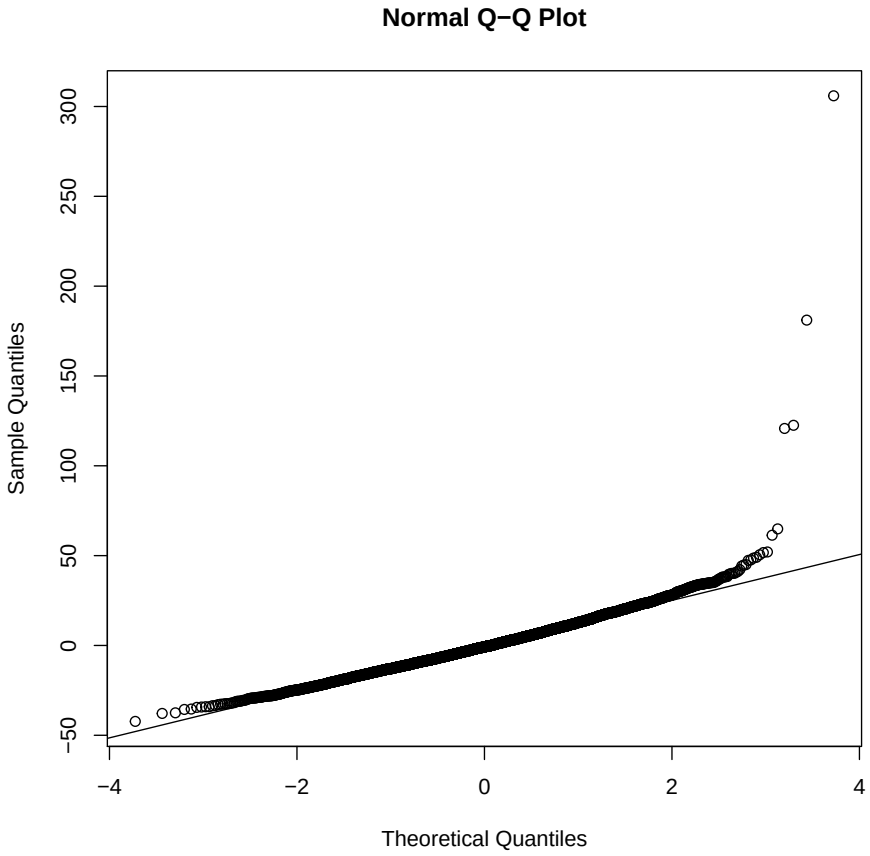
Q-Q plots for two data distributions With two data sets, say $x_1, \ldots x_n$ and y_1, y_m, one can compare their sample quantiles. The easiest way is when $n = m$, since then one just plots $x_{(i)}$ against $y_{(i)}$. (If $n \neq m$, one might pick a grid of p values, work out $\hat{F}_x^{-1}(p)$ and $\hat{F}_y^{-1}(p)$ by interpolation, and plot those against each other.) This should show points around the $x = y$ line when x and y are both drawn from the same distribution.

P-P plots An alternative to $Q-Q$ plots are $P-P$ plots, where both axes run from 0 to 1. Call the horizontal coordinate p_1 and the vertical coordinate p_2; what we plot is $p_2 = \hat{F}(F^{-1}(p_1))$. $F^{-1}(p_1)$ is the quantile which ought, under the distribution F, to be $\geq$ a fraction p_1 of the population; $\hat{F}(F^{-1}(p_1))$ shows the actual fraction of the sample which it exceeds. R doesn't have a built-in function to make this — can you write one?

Formal tests Comparing the histogram to a theoretical density can be formalized with a χ^2 test[5]. Checking whether the $Q-Q$ plot follows a straight line can be formalized in the Kolmogorov-Smirnov test. In both cases, since we're really estimating the a parameter of the reference distribution (the standard deviation), we need to take some care to account that in a hypothesis test. (A tweak to the K-S test which does this, when testing for Gaussianity, is the "Lilliefors test", which you can find in the package `nortest`.)

[4]When you call `qqnorm` in R, it actually goes through a calculation to find $\mathbb{E}\left[X_{(i)}\right]$ for $i \in 1:n$ under a Gaussian distribution, and plots *that* against the observed value $x_{(i)}$. As $n \to \infty$, $\mathbb{E}\left[X_{(i)}\right] \to \Phi^{-1}(i/n)$, but the difference can be important for small n, and/or i/n close to 0 or 1.

[5]This is why the visual comparison is informally called "χ by eye".




```
qqnorm(residuals(death.temp.lm))
qqline(residuals(death.temp.lm))
```

FIGURE 3.8: *QQ plot of the residuals, using the standard Gaussian distribution as the reference distribution.*

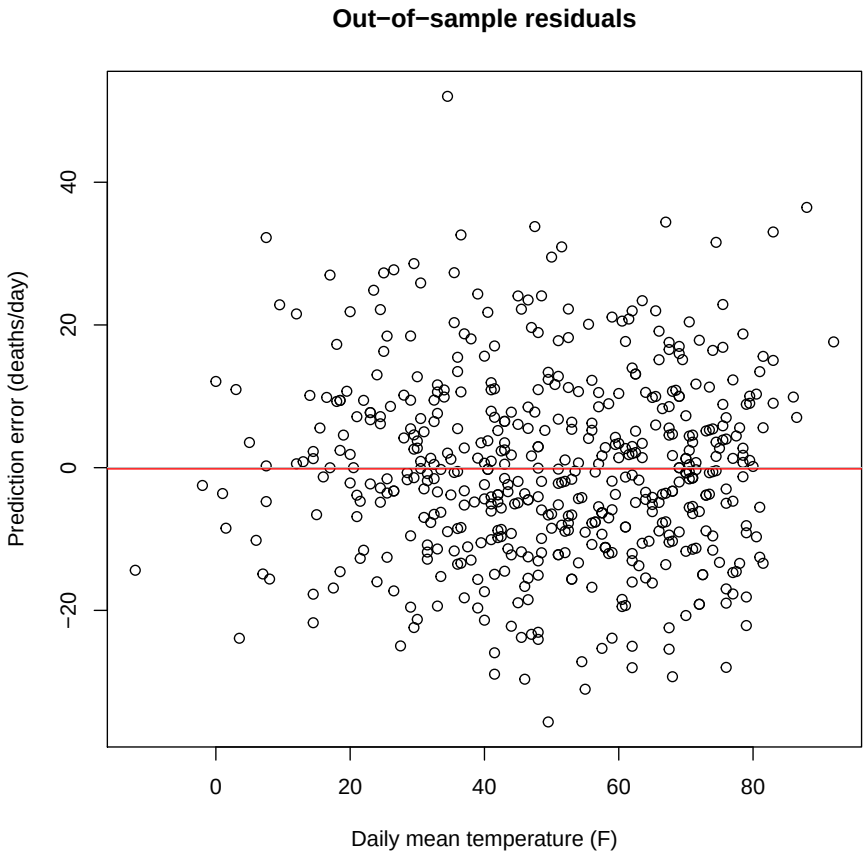
```
training.rows <- sample(1:nrow(chicago), size = round(nrow(chicago) * 0.9),
    replace = FALSE)
training.set <- chicago[training.rows, ]
testing.set <- chicago[-training.rows, ]
training.lm <- lm(death ~ tmpd, data = training.set)
testing.preds <- predict(training.lm, newdata = testing.set)
testing.residuals <- testing.set$death - testing.preds
```

FIGURE 3.9: *Code setting up a* random *division of the data into training and testing sets, and looking at how well the model does on points in the testing set (which it didn't get to see during estimation).*

3.1.6 Generalization

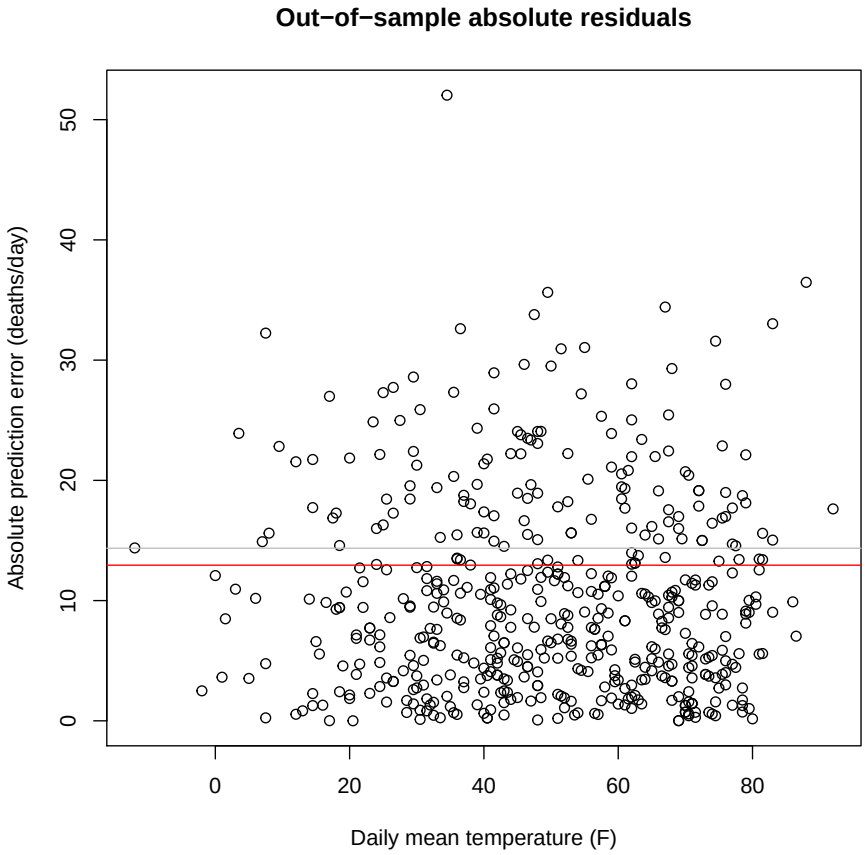
If the model assumptions are correct, it should be able to work about equally well on new data from the same source. Because the parameters were adjusted to fit the data we used to estimate the model, we should expect the prediction errors on new data to be slightly larger in magnitude, but they shouldn't be biased or otherwise show patterns. An important basic check on the model is therefore to divide the data into two parts, estimate the model on one part, the **training set**, and then examine the predictions and the residuals on the rest of the data, the **testing set**.

We can either make the division into training and testing sets by random sampling, or systematically. A random division ensures that the testing set has (almost) the same distribution as the training set. The averaged squared error on the testing set is therefore an unbiased estimate of the true mean squared error on new data. We will topic later in the course, under the heading of "cross-validation", since it is one of the most useful ways of selecting among competing models.




```
plot(testing.set$tmpd, testing.residuals, xlab = "Daily mean temperature (F)",
    ylab = "Prediction error (deaths/day)", main = "Out-of-sample residuals")
abline(h = 0, col = "grey")
abline(h = mean(testing.residuals), col = "red")
```

FIGURE 3.10: *Plot of residuals vs. temperature for the testing set. Remember that the data points here were* not *available to the model during estimation. The grey line marks the average we'd see on the training set (zero), while the red line shows the average on the testing set.*




```
plot(testing.set$tmpd, abs(testing.residuals), xlab = "Daily mean temperature (F)",
    ylab = "Absolute prediction error (deaths/day)", main = "Out-of-sample absolute residuals")
abline(h = sqrt(mean(residuals(training.lm)^2)), col = "grey")
abline(h = sqrt(mean(testing.residuals^2)), col = "red")
```

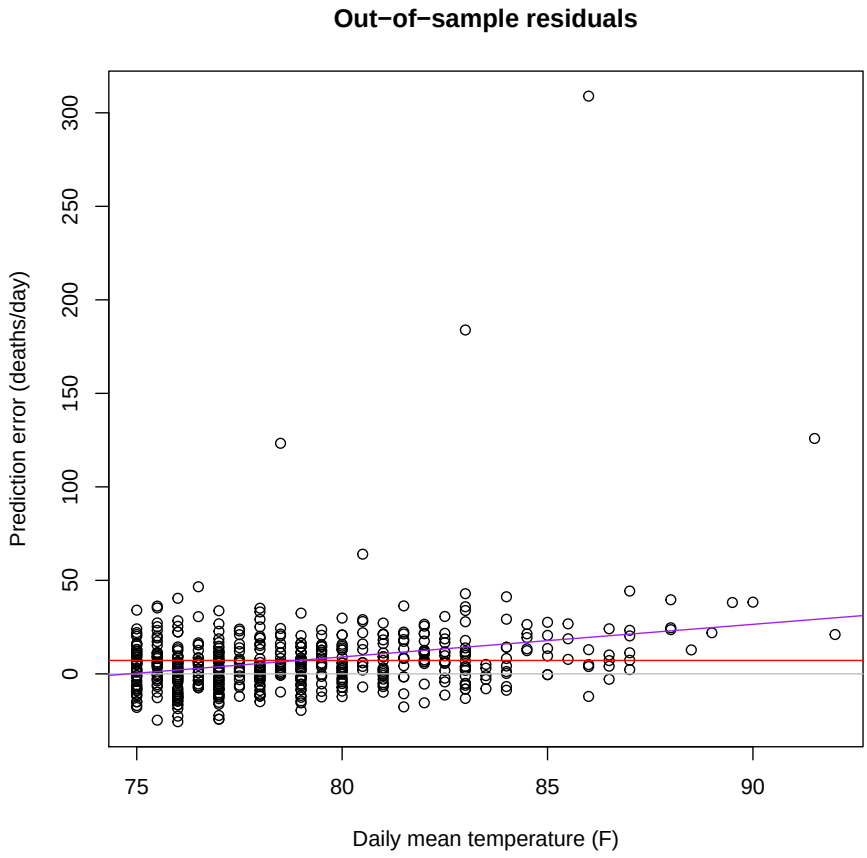
FIGURE 3.11: *As in Figure 3.10, but looking at the squared residuals.*

```
lowtemp.rows <- which(chicago$tmpd < 75)
lowtemp.set <- chicago[lowtemp.rows, ]
hightemp.set <- chicago[-lowtemp.rows, ]
lowtemp.lm <- lm(death ~ tmpd, data = lowtemp.set)
hightemp.preds <- predict(lowtemp.lm, newdata = hightemp.set)
hightemp.residuals <- hightemp.set$death - hightemp.preds
```

FIGURE 3.12: *Setting up a division of the data into a low-temperature training set and a high-temperature testing set.*

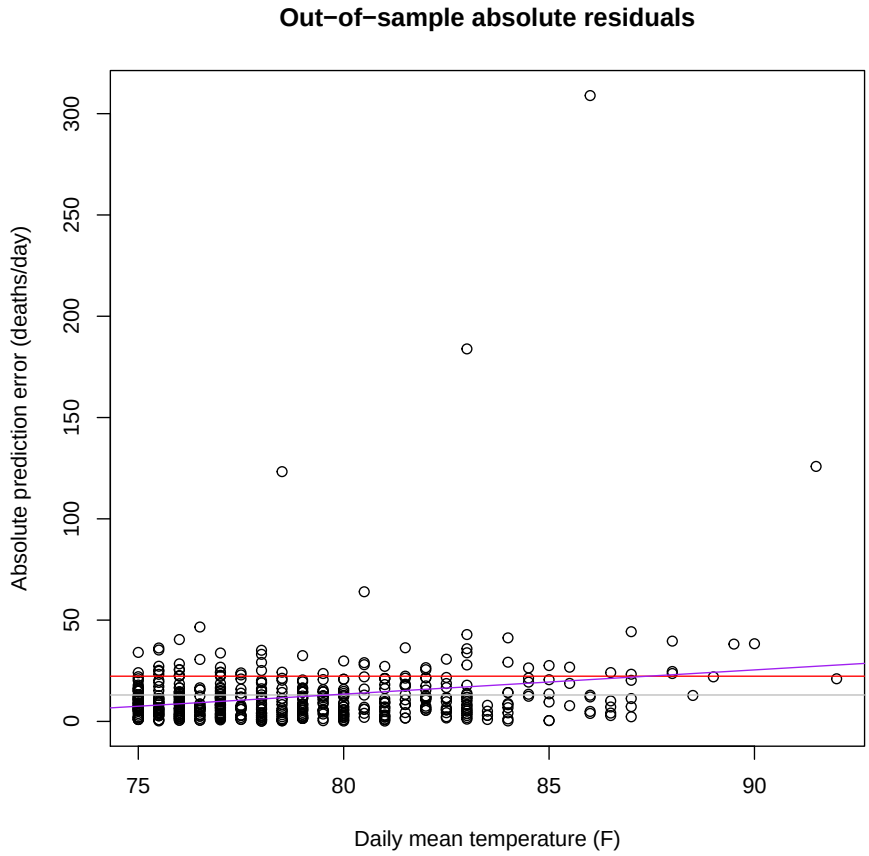
Extrapolative Generalization An alternative to random division, which can be even more useful for model checking, is to systematically make the testing set into a part of the data where the over-all fit of the model seems dubious based on other diagnostics. With our running examples, for instance, the model seems to do decently at lower temperatures, but starts to look iffy at high temperatures. We might, then, fit the model to low temperatures, and see whether it can extrapolate to high temperatures, or whether it seems to make systematic errors there. (We could also estimate the model on the high-temperature portion of the data, and see how well it extrapolates to the lower temperatures, or estimate on the middle range and see about both extremes, etc., etc.)

Generalize All the Things! *All* of the diagnostic plots discussed earlier can be combined with the trick of estimating the model on one part of the data, and then seeing whether it generalizes to the testing set. This is slightly more complicated than doing everything on the full data, but arguably has more power to detect problems with the model.




```
plot(hightemp.set$tmpd, hightemp.residuals, xlab = "Daily mean temperature (F)",
    ylab = "Prediction error (deaths/day)", main = "Out-of-sample residuals")
abline(h = 0, col = "grey")
abline(h = mean(hightemp.residuals), col = "red")
abline(lm(hightemp.residuals ~ hightemp.set$tmpd), col = "purple")
```

FIGURE 3.13: *Residuals vs. temperature, where the testing set here consists of days with temperature* ≥ 75 *degrees Farenheit, and the training set only those* < 75 *degrees. The grey line indicates the average residual we'd see on the training data (zero); the red line the average residual on the testing data; the purple a regression of residual on temperature, which ideally should have had slope zero.*




```
plot(hightemp.set$tmpd, abs(hightemp.residuals), xlab = "Daily mean temperature (F)",
    ylab = "Absolute prediction error (deaths/day)", main = "Out-of-sample absolute residuals")
abline(h = sqrt(mean(residuals(lowtemp.lm)^2)), col = "grey")
abline(h = sqrt(mean(hightemp.residuals^2)), col = "red")
abline(lm(abs(hightemp.residuals) ~ hightemp.set$tmpd), col = "purple")
```

FIGURE 3.14: *As in Figure 3.13, but for absolute residuals.*

3.2 Nonlinear Functions of X

When we plot the residuals against the predictor variable, we may see a curved or stepped pattern. This strongly suggests that the relationship between Y and X is not linear. At this point, it is often useful to fall back to, in fact, plotting the y_i against the x_i, and try to guess at the functional form of the curve.

3.2.1 Transformations

The easy case is when $Y = \beta_0 + \beta_1 f(x) + \epsilon$ for some fixed, easily-computed function f like $\sqrt{x}$, x^2, $\log x$, $\sin x$, etc. We then calculate $f(x_i)$, and run a simple linear regression of y_i on $f(x_i)$. The interpretation of the coefficients hardly changes, except that we need to replace X everywhere with $f(X)$ — $\beta_0 = \mathbb{E}[Y|f(X) = 0]$, β_1 is the difference $\mathbb{E}[Y|f(X) = f_0]$ and $\mathbb{E}[Y|f(X) = f_0 - 1]$, etc. This is called "transforming the predictor", and it only works this simply when the transformation itself doesn't have to be estimated, but can just be guessed at. Ideally, in fact, we derive the transformation from some physical (chemical, biological, psychological, sociological, economic, ...) theory. For instance, there are good reasons in physiology and psychology to say that an organism's behavioral response to a stimulus should vary with the logarithm of the stimulus's physical intensity[6]. A good check on such a transformation is to plot the y_i against the $f(x_i)$, and see that the data now fall on a straight line.

3.2.2 Nonlinear Least Squares

If the transformation does have to be estimated, but the functional form is known[7], then the method of least squares (or maximum likelihood) still applies. Taking the derivative of the mean squared error (or the log likelihood) with respect to the parameters and setting them equal to zero gives a set of normal or estimating equations. Usually, however, these equations are nonlinear, and don't have a closed-form solution. Finding the solution is thus called "solving a nonlinear least squares (NLS) problem". When we have theoretical reasons to use some thoroughly nonlinear model, say $Y = \beta_0 x^{\beta_1} + \epsilon$, we can still estimate it using NLS in this way[8].

3.2.3 Smoothing

An alternative to using a parametrized nonlinear model is to try to let the data tell us what the appropriate curve is — to take a "non-parametric" approach. The most basic sort of curve-fitting would just take a little interval around any

[6]This is also (roughly) true, at least in people, of the perceived intensity of the stimulus. Full sunlight carries $\approx 1000\text{W/m}^2$ of power, while the light of a full moon is $\approx 0.025\text{W/m}^2$, yet moonlight doesn't seem forty thousand times dimmer. The logarithmic relationship between perceived and physical intensity is sometimes called "Fechner's law" in psychophysics.

[7]For instance, if $Y = \beta_0 + \beta_1 \log(x + \beta_2) + \epsilon$, for unknown k.

[8]The relevant R commands are `optim`, which is a general-purpose minimization function, and `nls`, which is, unsurprisingly, specialized to nonlinear least squares.

point x, say from $x - h$ to $x + h$, and average all the y_i where x_i was in the interval:

$$\hat{m}(x) = \frac{\sum_{i=1}^{n} y_i I_{[x-h,x+h]}(x_i)}{\sum_{j=1}^{n} I_{[x-h,x+h]}(x_j)}$$

(Here $I_{[a,b]]}(x)$ is the **indicator function** for the interval $[a, b]$, i.e., 1 if $a \leq x \leq b$, and 0 otherwise.)

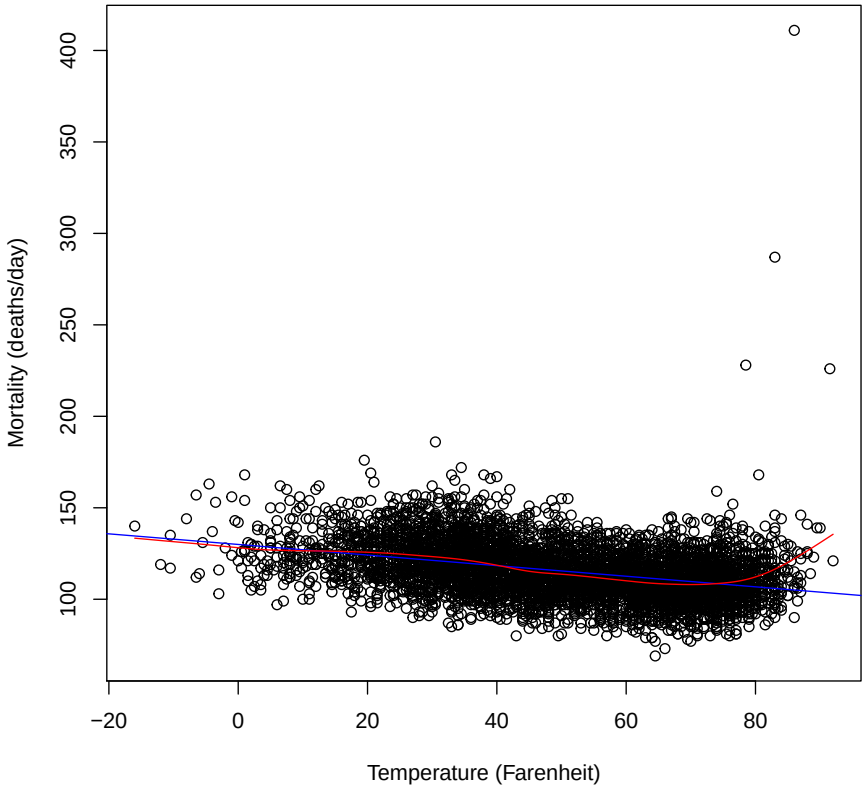
This sort of local averaging gives an $\hat{m}(x)$ which can make jerk steps as x changes. Another approach is **spline smoothing**: this looks for the function, called a **spline** which comes closest to the data points, subject to a constraint on the average curvature of the function. Allowing no curvature at all gives back the least squares line; allowing unlimited curvature gives a function which interpolates exactly between the data points[9]. Of course one has to decide how much curvature to allow; the best idea is generally to do what's called "cross-validation": hold back a little bit of the data, fit the spline with some level of curvature to the rest of the data, and see how well it predicts the held-back part; pick the curvature which generalizes best to unseen data points. While there is lot of R code for splines, because they're very useful, the most user-friendly is called `smooth.spline`.

```
smooth.spline(x, y, cv = TRUE)
```

returns an object which contains the estimated spline. (The default, `CV=FALSE`, is to use a fast approximation to cross-validation called "generalized cross-validation"; `CV=TRUE` is often a bit more accurate, if you can afford the time.) The commands `fitted` and `residuals` work on this object just like they do on `lm` objects; `predict` works very similarly, but the argument giving the new values must be a vector called `x`, not a data frame. If passed into the `lines` or `points` command, we get a plot of the fitted values.

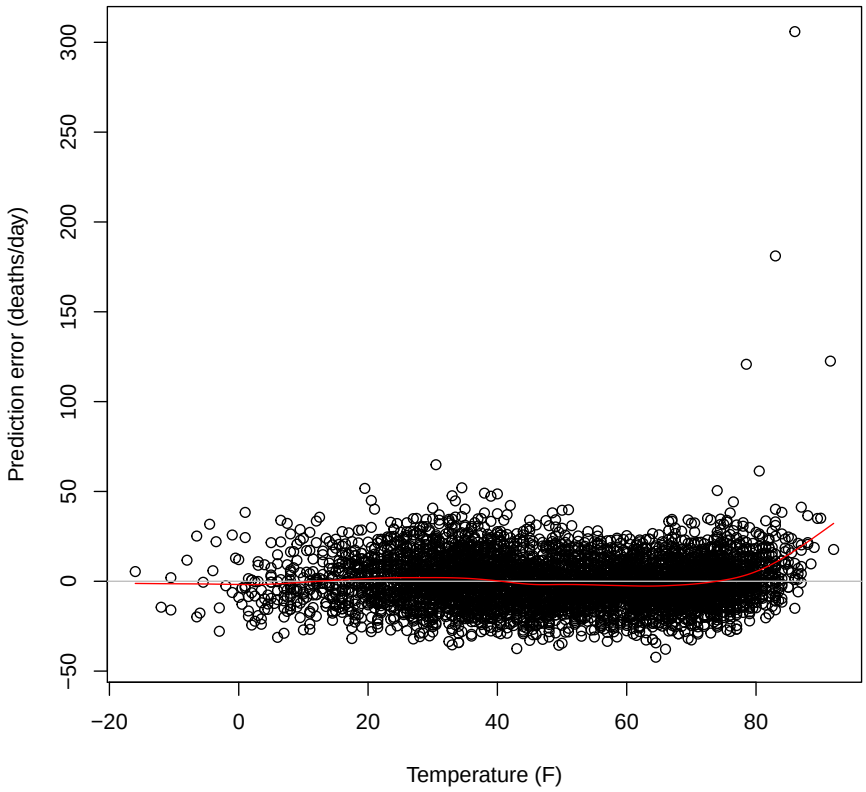
`smooth.spline` can also be used on the squared residuals (not the absolute residuals – why?) to try to get an estimate of the conditional variance, if you're pretty sure that the functional form of the regression is right.

[9]This turns out to be equivalent to a kind of weighted local averaging.



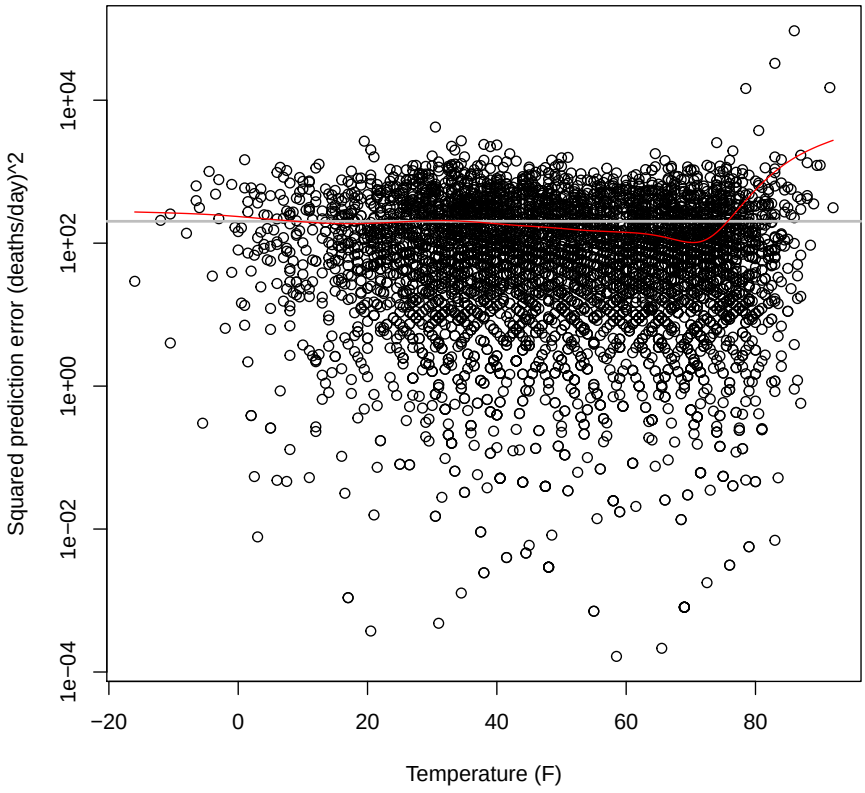

```
plot(death ~ tmpd, data = chicago, xlab = "Temperature (Farenheit)", ylab = "Mortality (deaths/d
death.temp.ss <- smooth.spline(x = chicago$tmpd, y = chicago$death, cv = TRUE)
abline(death.temp.lm, col = "blue")
lines(death.temp.ss, col = "red")
```

FIGURE 3.15: *Scatter-plot of mortality as a function of temperature, along with the estimated linear model (blue) and the estimated smoothing spline (red). Notice how the smoothing spline tracks the linear model over a wide range of temperatures, but then turns* up *for high temperatures.*




```
plot(chicago$tmpd, residuals(death.temp.lm), xlab = "Temperature (F)", ylab = "Prediction error (deaths/day)
abline(h = 0, col = "grey")
lines(smooth.spline(x = chicago$tmpd, y = residuals(death.temp.lm), cv = TRUE),
    col = "red")
```

FIGURE 3.16: *Estimating a smoothing spline on the residuals of the linear model. Ideally, the spline would be close to zero everywhere; here we see it's working pretty well,* except *at high temperature.*




```
plot(chicago$tmpd, residuals(death.temp.lm)^2, log = "y", xlab = "Temperature (F)",
    ylab = "Squared prediction error (deaths/day)^2")
abline(h = mean(residuals(death.temp.lm)^2), lwd = 2, col = "grey")
lines(smooth.spline(x = chicago$tmpd, y = residuals(death.temp.lm)^2, cv = TRUE),
    col = "red")
```

FIGURE 3.17: *Smoothing spline of squared residuals versus temperature. (Notice the logarithmic scale for the vertical axis, to compensate for the fact that some of the residuals are really big.) If we thought that the functional form of the regression was right, this would be a reasonable estimate of the conditional variance. Should we think that?*

3.3 Transforming the Response

Another way to try to accommodate nonlinearity is to transform the response variable, rather than the predictor. That is, one imagines the model is

$$g(Y) = \beta_0 + \beta_1 x + \epsilon_i$$

for some invertible function g. In more old-fashioned sources, like our textbook, this is advocated as a way of handling non-constant variance, or non-Gaussian noise[10]. A better rationale is that it might in fact be true. Since the transformation g has an inverse, we can write

$$Y = g^{-1}\left(\beta_0 + \beta_1 x + \epsilon_i\right)$$

Even if $\epsilon_i \sim N(0, \sigma^2)$, this implies hat Y will have a non-Gaussian distribution, with a non-linear relationship between $\mathbb{E}[Y|X = x]$ and x, and a non-constant variance. If that's actually the case, we'd like to incorporate that into the model. For instance, Y might be an integer-valued count variable, or even a binary-valued categorical variable, and then we pretty much have to have some sort of non-Gaussian noise in Y. (Indeed, the noise around $\mathbb{E}[Y|X = x]$ isn't even strictly additive.)

Let me illustrate these points by working through a log transformation of Y. Suppose

$$\log Y = \beta_0 + \beta_1 x + \epsilon$$

where $\epsilon \sim N(0, \sigma^2)$, independent of x. The inverse function to log is exp, so this is logically equivalent to

$$Y = e^{\beta_0 + \beta_1 x} e^{\epsilon} = e^{\beta_0 + \beta_1 x} \xi$$

where $\xi = e^{\epsilon}$ follows a **log-normal** distribution, with $\mathbb{E}[\log \xi] = 0$, $\mathrm{Var}[\log \xi] = \sigma^2$. One can show[11] that $\mathbb{E}[\xi] = e^{\sigma^2/2}$, that $\mathrm{Var}[\xi] = (e^{\sigma^2} - 1)e^{\sigma^2}$, that the median of ξ is 1, and that ξ is, consequently, skewed to the right, with an asymmetric distribution. Conversely, if $Y = e^{\beta_0 + \beta_1 x} + \epsilon$, with $\epsilon \sim N(0, \sigma^2)$, then it is *not* the case that $\log Y = \beta_0 + \beta_1 x + \eta$, for some Gaussian η.

The point of this is that transforming the response thoroughly changes the interpretation of the parameters: β_0 is not $\mathbb{E}[Y|X = x]$, but $\mathbb{E}[g(Y)|X = x]$, β_1 is the slope of $\mathbb{E}[g(Y)|X = x]$, etc. It also implies very particular, even peculiar, models for noise around the regression line. This makes it impossible to compare mean squared errors or log-likelihoods before and after transformations[12]. One can, however, look at the residuals for the *transformed* response, and see whether they are flat in the predictor variable, etc.

[10] To be quite honest, this seems like as clear a case of being enslaved by one's own tools as one could hope to find in the sciences. More charitably: it made a certain amount of sense before people figures out how to deal with non-constant variance (around 1935) and non-Gaussian noise (around 1980).

[11] Check Wikipedia if you don't believe me.

[12] More exactly, log-likelihoods *can* be compared, but we need to compensate for the transformation.

Log transformations with log-normal multiplicative noise The usual argument for anticipating Gaussian noise is the central limit theorem: if lots of little disturbing causes add up their contributions to the response variable, and they're roughly independent and of comparable size, we'll expect that their net effect is Gaussian. If instead of adding up they multiply together, however, the central limit theorem does not apply directly. Since logarithms turn multiplication into addition, when effects multiply we may anticipate log-normal fluctuations. If the simple linear regression model applies after taking the log of everything,

$$\begin{aligned} \log Y &= \beta_0 + \beta_1 log x + \epsilon && (3.10)\\ \epsilon &\sim N(0, \sigma^2) && (3.11)\\ \epsilon &\text{ independent of } X && (3.12) \end{aligned}$$

then, undoing the log,

$$\begin{aligned} Y &= e^{\beta_0} x^{\beta_1} \eta && (3.13)\\ \eta &\sim \log N(0, \sigma^2) \\ \eta &\text{ independent of } X \end{aligned}$$

This sort of model is not actually unheard of in practice, because there really are situations where causes multiply.

Eq. 3.13 is a *different model* from

$$Y = e^{\beta_0} x^{\beta_1} + \epsilon$$

We would estimate the formal model by doing a linear regression of $\log Y$ on $\log X$ with Gaussian noise. We would estimate the latter model by nonlinear least squares. xs

3.4 Looking Forward

Non-constant variance in a linear model is actually comparatively easy to handle, if we can work out what variance is: the trick is a modification of the method of least squares called "weighted least squares", which we will cover later in the course. Similarly, correlated noise can be handled through a modification called "generalized least squares", which we'll also cover. There are a range of simulation-based techniques for doing inference when the Gaussian-noise assumption fails; these have opaque, forcedly-whimsical names like "the jackknife" and "the bootstrap", and we'll get to these towards the end of the course.

The easiest direction to go in is to add more predictor variables, and hope that the response is linear in each of them.

3.5 Residuals under Misspecification

(This section is optional, but strongly recommended.)

Suppose that the conditional expectation function $\mu(x) \equiv \mathbb{E}[Y|X=x]$ is not linear in x, but we use a linear model anyway. We know that there is a best linear approximation to this μ, with slope $\beta_1 = \mathrm{Cov}[X,Y]/\mathrm{Var}[X]$ and intercept $\beta_0 = \mathbb{E}[Y] - \beta_1 \mathbb{E}[X]$. So we can write

$$Y = \mu(x) + \epsilon \tag{3.14}$$

$$Y = \beta_0 + \beta_1 x + (\mu(x) - \beta_0 + \beta_1 x) + \epsilon \tag{3.15}$$

It's annoying to keep writing out $\mu(x) - \beta_0 + \beta_1 x$, so let's define that to be a new function, $\nu(x) \equiv \mu(x) - \beta_0 + \beta_1 x$. You can show that

$$\mathbb{E}[\nu(X)] = 0$$

and that

$$\mathrm{Cov}[\nu(X), X] = 0$$

However, it is not in general the case that

$$\mathbb{E}[\nu(X)|X=x] = 0$$

Suppose now that — through ignorance or as a deliberate strategy — we fit a linear model by least squares. It will be the case that

$$Y_i = \beta_0 + \beta_1 x_i + \eta_i$$

where η_i has expectation zero and is uncorrelated with X. But what we are treating as noise is partially the real noise, and partially the systematic effect of ignoring the nonlinearities:

$$\eta_i = \epsilon_i + \nu(x_i)$$

Thus, what looks like noise to the linear model will *not* have conditional expectation zero:

$$\mathbb{E}[\eta|X=x] = \nu(x)$$

The conditional variance doesn't alter,

$$\mathrm{Var}[\eta|X=x] = \mathrm{Var}[\epsilon|X=x]$$

but it's no longer equal to the expected square:

$$\mathbb{E}\left[\eta^2|X=x\right] = \nu^2(x) + \mathrm{Var}[\epsilon|X=x]$$

When we run least squares, our slope and intercept will still tend to converge on those of the optimal linear model. We can even still use the expressions we worked out for $\hat{\beta}_0$ (or $\hat{\beta}_1$) in terms of β_0 (or β_1) and a weighted sum of random variables — but now those are the η_i, not the ϵ_i. This implies that conditional expectation of the residuals will *not* be zero (in general), the conditional expectation of the squared residuals will not be constant, and so forth.

Chapter 4

Linear regression: Matrix formulation

4.1 Simple Linear regression

We come back to the simple linear regression model we studied in the first 2 chapters and we reformulate it in matrix form. Our data consist of n observations of the predictor variable X and the response variable$Y$, i.e, $(x_1, y_1), \ldots, (x_n, y_n)$. We wish to fit the model

$$Y = \beta_0 + \beta_1 X + \epsilon \tag{4.1}$$

where $\mathbb{E}[\epsilon|X = x] = 0$, $\mathrm{Var}(\epsilon|X = x) = \sigma^2$, and ϵ is uncorrelated across the measurements.

We introduce vectors and matrices associated to our model

$$\mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix}, \quad \beta = \begin{bmatrix} \beta_0 \\ \beta_1 \end{bmatrix}, \quad \mathbf{x} = \begin{bmatrix} 1 & x_1 \\ 1 & x_2 \\ \vdots & \vdots \\ 1 & x_n \end{bmatrix}, \quad \epsilon = \begin{bmatrix} \epsilon_1 \\ \epsilon_2 \\ \vdots \\ \epsilon_n \end{bmatrix} \tag{4.2}$$

The matrix $\mathbf{x}$ is sometimes called the design matrix, ϵ the noise vector and β the regression vector. $\mathbf{x}\beta$ is the following n-dimensional vector

$$\mathbf{x}\beta = \begin{bmatrix} \beta_0 + \beta_1 x_1 \\ \beta_0 + \beta_1 x_2 \\ \vdots \\ \beta_0 + \beta_1 x_n \end{bmatrix} \tag{4.3}$$

Then, (4.1) can be rewritten in matrix form as

$$\mathbf{y} = \mathbf{x}\beta + \epsilon. \tag{4.4}$$

Least Squares Estimator. Using β as our regression vector. the associated vector of prediction errors is given by

$$\mathbf{e}(\beta) = \mathbf{y} - \mathbf{x}\beta. \tag{4.5}$$

The mean square error at β is given by

$$MSE(\beta) = \frac{1}{n}\sum_{i=1}^{n} e_i^2(\beta) = \frac{1}{n}\|\mathbf{e}\|^2 = \frac{1}{n}\|\mathbf{y} - \mathbf{x}\beta\|^2.$$

The least squares estimator satisfies

$$\hat{\beta} = \text{argmin}_\beta \frac{1}{n}\|\mathbf{e}\|^2 = \frac{1}{n}\|\mathbf{y} - \mathbf{x}\beta\|^2. \tag{4.6}$$

Solving the above minimization problem requires to compute the gradient of $MSE(\cdot)$

$$\nabla MSE(\beta) = \frac{2}{n}(\mathbf{x}^\top\mathbf{x}\beta - \mathbf{x}^\top\mathbf{y}).$$

We find the critical points, that is the β that satisfy $\nabla MSE(\beta) = 0$. We obtain the set of normal equations:

$$\mathbf{x}^\top\mathbf{x}\hat{\beta} - \mathbf{x}^\top\mathbf{y} = 0.$$

Next, we need to check that the critical points are indeed minimizers. In that case, this is straightforward as the MSE function is convex.

Assuming that $\mathbf{x}^\top\mathbf{x}$ is nonsingular, the solution to the set of normal equations is unique :

$$\hat{\beta} = (\mathbf{x}^\top\mathbf{x})^{-1}\mathbf{x}^\top\mathbf{y} \tag{4.7}$$

$$= (\mathbf{x}^\top\mathbf{x}/n)^{-1}\frac{1}{n}\mathbf{x}^\top\mathbf{y}. \tag{4.8}$$

Note that

$$\mathbf{x}^\top\mathbf{x}/n = \begin{pmatrix} 1 & \bar{x} \\ \bar{x} & \bar{x^2} \end{pmatrix}, \quad \frac{1}{n}\mathbf{x}^\top\mathbf{y} = \begin{pmatrix} \bar{y} \\ \bar{xy} \end{pmatrix}.$$

The inverse of the 2×2 matrix $\mathbf{x}^\top\mathbf{x}/n$ is given by

$$(\mathbf{x}^\top\mathbf{x}/n)^{-1} = \frac{1}{\bar{x^2} - (\bar{x})^2}\begin{pmatrix} \bar{x^2} & -\bar{x} \\ -\bar{x} & 1 \end{pmatrix}.$$

Thus, we obtain

$$\begin{pmatrix} \hat{\beta}_0 \\ \hat{\beta}_1 \end{pmatrix} = \begin{pmatrix} \bar{y} - \hat{\beta}_1\bar{x} \\ \frac{\bar{xy} - \bar{x}\bar{y}}{\bar{x^2} - \bar{x}^2} \end{pmatrix}.$$

We recognize the formulas obtained in Chapter 1 for the simple linear regression model. Note however that the formula (4.7) is true for multiple linear regression with an arbitrary number of regressors provided that the matrix $\mathbf{x}^\top\mathbf{x}$ is nonsingular.

Fitted values and Residuals. The vector of fitted values $\overline{\mathbf{m}(\mathbf{x})}$ is

$$\hat{\mathbf{m}} = \overline{\mathbf{m}(\mathbf{x})} = \mathbf{x}\hat{\beta}. \tag{4.9}$$

Using our equation for $\hat{\beta}$, we have

$$\hat{\mathbf{m}} = \mathbf{x}(\mathbf{x}^\top\mathbf{x})^{-1}\mathbf{x}^\top\mathbf{y} = \mathbf{H}\mathbf{y}, \tag{4.10}$$

where

$$\mathbf{H} = \mathbf{x}(\mathbf{x}^\top\mathbf{x})^{-1}\mathbf{x}^\top. \tag{4.11}$$

Interestingly, if we obtain new observations of the model at the same $\mathbf{x}$, then we can use $\mathbf{m} = \mathbf{H}\mathbf{y}$ to obtain the fitted values for the new observations. The matrix $\mathbf{H}$ is called the **influence matrix** (or sometimes the **hat matrix**).

1. *Influence.* we have $\frac{\partial \hat{m}_i}{\partial y_j} = H_{ij}$. Thus, H_{ij} is the rate at which the i^{th} fitted value changes as we vary the j^{th} observation, the influence that observation has on that fitted value.
2. *Projection.* The matrix $\mathbf{H}$ corresponds to the orthogonal projection onto the range of $\mathbf{x}$.

Reminder on geometry and projections . A linear application P is an orthogonal projection onto its range if and only if it is self-adjoint $P^\top = P$ and idempotent $P^2 = P$. The projection of vector $\mathbf{u}$ onto the line generated by vector $\mathbf{v}$ is given by

$$P(\mathbf{u}) = \frac{\mathbf{v}\mathbf{v}^\top}{\|v\|^2}\mathbf{u}.$$

If $\mathbf{u}$ belongs already to $l.s.(\mathbf{v})$, then $P(\mathbf{u}) = \mathbf{u}$. That means that for any vector $\mathbf{u}$, $P(\mathbf{u}) \in l.s.(\mathbf{v})$ and thus $P(P(\mathbf{u})) = P(\mathbf{u})$. That is $P^2 = P$.

The orthogonal projection P_M onto a linear subspace M of $\mathbb{R}^n$ dimension k can be defined as follows. Take an orthonormal basis $\mathbf{v}_1, \ldots, \mathbf{v}_k$ of M, then

$$P_M = \sum_{j=1}^{k} \mathbf{v}_j\mathbf{v}_j^\top.$$

The influence matrix provides a matrix representation of the orthogonal projection onto the range of $\mathbf{x}$ provided that $\mathbf{x}$ is full-rank, which implies indeed that $\mathbf{x}^\top\mathbf{x}$ is non-singular.

residuals. we recall that the vectors of residuals is given by

$$\mathbf{e} = \mathbf{y} - \mathbf{x}\hat{\beta} = \mathbf{y} - \mathbf{H}\mathbf{y} = (I - \mathbf{H})\mathbf{y}.$$

The influence $\partial \mathbf{e}_i / \partial y_j = (I - \mathbf{H})_{ij}$. The rate at which the residual $\mathbf{e}_i$ varies with observation $\mathbf{y}_j$ is given by $(I - \mathbf{H})_{ij}$. Note that $(I - \mathbf{H})$ is the orthogonal projection onto range$(\mathbf{x})^\perp$.

Thus, we have

$$MSE(\hat{\beta}) = \frac{1}{n}\mathbf{y}^\top(I - \mathbf{H})^\top(I - \mathbf{H})\mathbf{y} = \frac{1}{n}\mathbf{y}^\top(I - \mathbf{H})\mathbf{y}. \tag{4.12}$$

Expectation and covariance. We have $\mathbf{y} = \mathbf{x}\beta + \epsilon$. We observe first that the vector of fitted values satisfies

$$\hat{m} = \mathbf{H}\mathbf{y} = \mathbf{x}\beta + \mathbf{H}\epsilon,$$

since $\mathbf{H}$ is the orthogonal projection onto range($\mathbf{x}$) and we obviously have that $\mathbf{x}\beta \in$ range($\mathbf{x}$).

Thus, we get

$$\mathbb{E}[\hat{m}] = \mathbf{x}\beta, \tag{4.13}$$

and

$$\mathrm{Cov}(\hat{m}) = \mathrm{Cov}(\mathbf{H}\mathbf{y}) = \mathbf{H}\mathrm{Cov}(\epsilon)\mathbf{H}^\top = \sigma^2\mathbf{H}I\mathbf{H}^\top = \sigma^2\mathbf{H}. \tag{4.14}$$

Similarly, we obtain for the vector of residuals

$$\mathbf{e} = (I - \mathbf{H})\mathbf{y} = (I - \mathbf{H})\epsilon.$$

Thus its expectation is 0 and its covariance is

$$\mathrm{Cov}(\mathbf{e}) = \mathrm{Cov}((I-\mathbf{H})\epsilon) = (I-\mathbf{H})\mathrm{Cov}(\epsilon)(I-\mathbf{H})^\top = \sigma^2(I-\mathbf{H})I(I-\mathbf{H})^\top = \sigma^2(I-\mathbf{H}). \tag{4.15}$$

Recall that the expected MSE is given by $\mathbb{E}[\frac{1}{n}\|\mathbf{e}\|^2]$. In view of the previous display, $\|\mathbf{e}\|^2/\sigma^2$ follows a χ^2 distribution with $\mathrm{tr}(I - \mathbf{H}) = n - 2$ degrees of freedom. Thus we get

$$\mathbb{E}[\frac{1}{n}\|\mathbf{e}\|^2] = \sigma^2\frac{n-2}{n}. \tag{4.16}$$

Sampling distribution. So far, all the properties we have deduced in this chapter do not require any conditions on the distribution noise other than $\mathbb{E}[\epsilon] = 0$ and $\mathrm{Cov}(\epsilon) = \sigma^2 I$. Now we assume that the noise variables are i.i.d. gaussian random variables. We have

$$\hat{\beta} = (\mathbf{x}^\top\mathbf{x})^{-1}\mathbf{x}^\top\mathbf{y} = \beta + (\mathbf{x}^\top\mathbf{x})^{-1}\mathbf{x}^\top\epsilon. \tag{4.17}$$

Since ϵ is a Gaussian vector, so is $(\mathbf{x}^\top\mathbf{x})^{-1}\mathbf{x}^\top\epsilon$. We compute its expectation and covariance:

$$\mathbb{E}[(\mathbf{x}^\top\mathbf{x})^{-1}\mathbf{x}^\top\epsilon] = (\mathbf{x}^\top\mathbf{x})^{-1}\mathbf{x}^\top\mathbb{E}[\epsilon] = 0, \tag{4.18}$$

and

$$\mathrm{Cov}(\mathbf{x}^\top\mathbf{x})^{-1}\mathbf{x}^\top\epsilon) = (\mathbf{x}^\top\mathbf{x})^{-1}\mathbf{x}^\top\mathrm{Cov}(\epsilon)\mathbf{x}((\mathbf{x}^\top\mathbf{x})^{-1})^\top \tag{4.19}$$

$$= \sigma^2(\mathbf{x}^\top\mathbf{x})^{-1}\mathbf{x}^\top\mathbf{x}((\mathbf{x}^\top\mathbf{x})^{-1})^\top \tag{4.20}$$

$$= \sigma^2(\mathbf{x}^\top\mathbf{x})^{-1}. \tag{4.21}$$

Thus

$$\hat{\beta} \sim N(\beta, \sigma^2(\mathbf{x}^\top\mathbf{x})^{-1}). \tag{4.22}$$

We recall that

$$\left(\frac{1}{n}\mathbf{x}^\top\mathbf{x}\right)^{-1} = \frac{1}{\overline{x^2} - (\bar{x})^2}\begin{pmatrix} \overline{x^2} & -\bar{x} \\ -\bar{x} & 1 \end{pmatrix}.$$

We deduce from the previous display that

$$\mathrm{Cov}(\hat{\beta}_0, \hat{\beta}_1) = -\frac{\sigma^2}{n}\frac{\bar{x}}{\overline{x^2} - \bar{x}^2}, \quad \mathrm{Var}(\hat{\beta}_0) = \frac{\sigma^2}{n}(1 + \frac{\bar{x}^2}{s_x^2}, \quad \mathrm{Var}(\hat{\beta}_0) = \frac{\sigma^2}{n s_x^2},$$

with $s_x^2 = \overline{x^2} - \bar{x}^2$.

We leave it as an exercise to determine the distributions of the residuals under the gaussian assumptions. It follows from similar arguments as those used above to determine the distribution of $\hat{\beta}$.

4.2 Multiple Linear Regression

We consider the following model with p predictors $X_1, \ldots, X_p$ and response y.

$$y = \beta_0 + \sum_{j=1}^{p} \beta_j X_j + \epsilon, \tag{4.23}$$

where the regression parameter $\beta = (\beta_0, \beta_1, \ldots, \beta_p)^\top \in \mathbb{R}^{p+1}$ is unknown. We have n i.i.d. observations $(X_{i,1}, \ldots, X_{i,p}, y_i)$ following (4.23):

$$y_i = \beta_0 + \sum_{j=1}^{p} \beta_j X_{j,i} + \epsilon_i. \tag{4.24}$$

Set

$$\mathbf{y} = (y_1, \cdots, y_n)^\top \in \mathbb{R}^n, \quad \epsilon = (\epsilon_1, \cdots, \epsilon_n)^\top \in \mathbb{R}^n, \quad \mathbf{x} = (X_{i,j-1})_{1 \le i \le n, 1 \le j \le p+1} \in \mathbb{R}^{n \times (p+1)},$$

with the convention $x_{i,0} = 1$ for any $1 \le i \le n$.

Then, the matrix reformulation of (4.24) is

$$\mathbf{y} = \mathbf{X}\beta + \epsilon, \tag{4.25}$$

One advantage of the matrix formulation is that the formulas (4.6)-(4.22) we derive in the simple linear model hold true for the multiple regression model without any modifications.

In particular, the LSE is

$$\hat{\beta} = (\mathbf{x}^\top \mathbf{x})^{-1} \mathbf{x}^\top \mathbf{y},$$

provided that $\mathbf{x}^\top \mathbf{x}$ is non-singular. Equivalently, the predictors (columns of $\mathbf{x}$) are linearly independent.

The fitted values (i.e., estimates of the conditional means at data points used to estimate the model) are given by the "hat" or "influence" matrix:

$$\widehat{\mathbf{m}} = \mathbf{x}\hat{\beta} = \mathbf{H}\mathbf{y} \tag{4.26}$$

which is symmetric and idempotent. The residuals are given by

$$\mathbf{e} = (\mathbf{I} - \mathbf{H})\mathbf{y} \tag{4.27}$$

and $\mathbf{I} - \mathbf{H}$ is the orthogonal projector onto the range of $\mathbf{x}$.

The expected mean squared error has a small negative bias:

$$\mathbb{E}\left[\hat{\sigma}^2\right] = \mathbb{E}\left[\frac{1}{n}\mathbf{e}^T\mathbf{e}\right] = \sigma^2\frac{n-p-1}{n} = \sigma^2\left(1 - \frac{p+1}{n}\right) \tag{4.28}$$

Since $\mathbf{H}\mathbf{x}\beta = \mathbf{x}\beta$, the residuals can also be written

$$\mathbf{e} = (\mathbf{I} - \mathbf{H})\epsilon \tag{4.29}$$

hence

$$\mathbb{E}\left[\mathbf{e}\right] = \mathbf{0} \tag{4.30}$$

and

$$\mathrm{Var}\left[\mathbf{e}\right] = \sigma^2(\mathbf{I} - \mathbf{H}) \tag{4.31}$$

Under the Gaussian noise assumption $\epsilon \sim N(0, \sigma^2 I)$, $\hat{\beta}$, $\hat{\mathbf{m}}$ and $\mathbf{e}$ all have Gaussian distributions.

4.2.1 Point Predictions

Say that $\mathbf{x}'$ is the $m \times (p+1)$ dimensional matrix storing the values of the predictor variables at m points where we want to make predictions. (These may or may not include points we used to estimate the model, and m may be bigger, smaller or equal to n.) Similarly, let $\mathbf{Y}'$ be the $m \times 1$ matrix of random values of Y at those points. The point predictions we want to make are

$$\mathbb{E}\left[\mathbf{Y}'|\mathbf{X}' = \mathbf{x}'\right] = \mathbf{m}(\mathbf{x}') = \mathbf{x}'\beta \tag{4.32}$$

and we *estimate* this by

$$\widehat{\mathbf{m}}(\mathbf{x}') = \mathbf{x}'\widehat{\beta} \tag{4.33}$$

which is to say

$$\widehat{\mathbf{m}}(\mathbf{x}') = \mathbf{x}'(\mathbf{x}^T\mathbf{x})^{-1}\mathbf{x}^T\mathbf{y} \tag{4.34}$$

(It's easy to verify that when $\mathbf{x}' = \mathbf{x}$, this reduces to $\mathbf{H}\mathbf{y}$.)

Notice that the point predictions we make *anywhere* are always weighted sums (linear combinations) of the values of the response we happened to observe when we estimated the model. The weights just depend on the values of the predictors at the original data points, and at the points where we'll be making predictions.

4.3 Multiple linear regression in R

It works pretty much the same as for the simple linear model.

```
mobility <- read.csv("~/Dropbox/Cours Polytechnique/regression XHEC regression/data/mobility.csv")
```

The only real change is that we need to tell lm, through the formula, what all the predictor variables are:

```
mob.lm <- lm(Mobility ~ Commute + Latitude + Longitude, data = mobility)
```

The order of the predictor variables only matters for the order in which the coefficients will be listed. All of the utility functions we already know still work, in exactly the same way:

```
print(mob.lm)
##
## Call:
## lm(formula = Mobility ~ Commute + Latitude + Longitude, data = mobility)
##
## Coefficients:
## (Intercept)      Commute     Latitude    Longitude
##  -3.136e-02    2.010e-01    9.383e-04   -4.305e-05


coefficients(mob.lm)
##   (Intercept)       Commute      Latitude     Longitude
## -3.136000e-02  2.009679e-01  9.383055e-04 -4.304546e-05


confint(mob.lm)
##                      2.5 %        97.5 %
## (Intercept) -0.0563094963 -0.0064104992
## Commute      0.1738437953  0.2280920687
## Latitude     0.0003580771  0.0015185339
## Longitude   -0.0002827799  0.0001966889


head(fitted(mob.lm))
##          1          2          3          4          5          6
## 0.07172344 0.06156703 0.07867982 0.06006085 0.06464329 0.06943562


head(residuals(mob.lm))
##            1            2            3            4            5
## -0.009524631 -0.007915094 -0.006044680 -0.003779634 -0.019842494
##            6
## -0.017599771
```

`summary` provides all these informations at once and also the results of differents tests:

```
summary(mob.lm)
##
## Call:
## lm(formula = Mobility ~ Commute + Latitude + Longitude, data = mobility)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -0.17583 -0.02222 -0.00586  0.01758  0.32290
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept) -3.136e-02  1.271e-02  -2.468  0.01383 *
## Commute      2.010e-01  1.382e-02  14.546  < 2e-16 ***
## Latitude     9.383e-04  2.956e-04   3.175  0.00156 **
## Longitude   -4.305e-05  1.221e-04  -0.353  0.72456
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.04227 on 725 degrees of freedom
## Multiple R-squared:  0.3583,Adjusted R-squared:  0.3557
## F-statistic: 134.9 on 3 and 725 DF,  p-value: < 2.2e-16
```

`predict` also works in exactly the same way, only we need to give a data frame with columns for each of the predictor variables:

```
predict(mob.lm, newdata = data.frame(Commute = 0.5, Latitude = 40.35, Longitude = -79.92))
##         1
## 0.1104248
```

Sometimes it could be useful to draw a bivariate scatter-plot for every pair of variables. It can be done in R through the command `pairs` (Figure **??**).

4.4 Tests and Confidence Sets for Multiple Coefficients

We consider the Gaussian-noise multiple linear regression model

$$Y = \beta_0 + \beta_1 X_1 + \ldots + \beta_p X_p + \epsilon \tag{4.35}$$

with $\epsilon \sim N(0, \sigma^2)$ independent of the X_is and independent across observations.

We recall the least squares or maximum likelihood estimators of the slopes are given by,

$$\widehat{\beta} = (\mathbf{x}^T\mathbf{x})^{-1}\mathbf{x}^T\mathbf{y} \tag{4.36}$$

Under these assumptions, the estimator has a multivariate Gaussian distribution,

$$\widehat{\beta} \sim N(\beta, \sigma^2(\mathbf{x}^T\mathbf{x})^{-1}) \tag{4.37}$$

```
pairs(~Mobility + Commute + Latitude + Longitude, data = mobility)
```

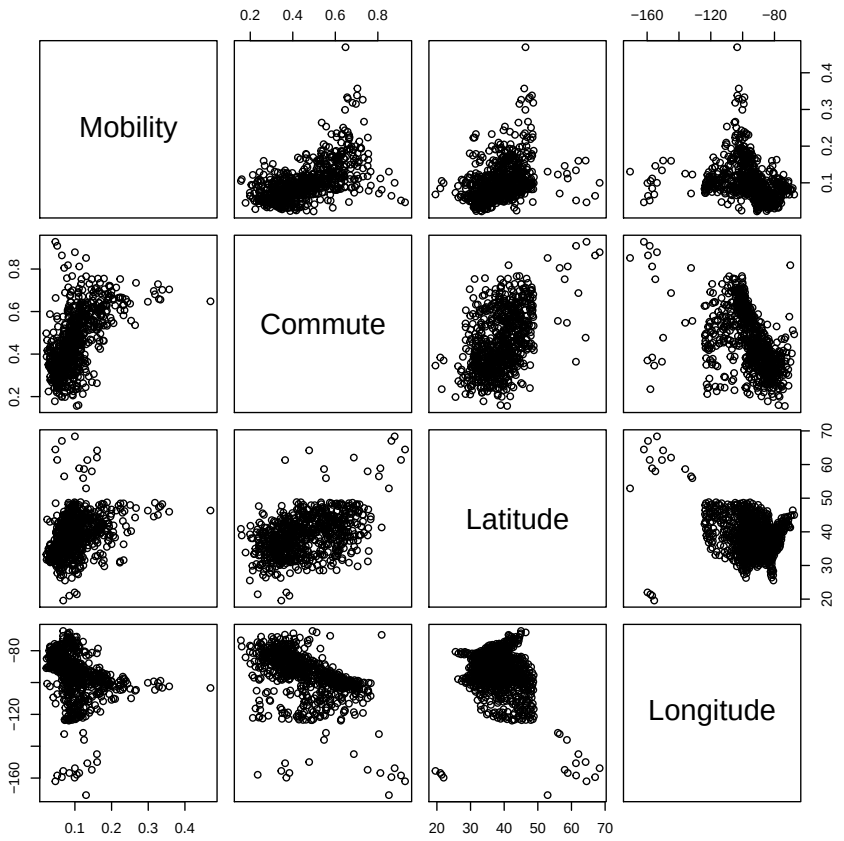


FIGURE 4.1: *Example of using pairs: the formula has an empty left-hand side (because there isn't really a distinguished response variable), and all the variables we want to plot on the right-hand side. If we left out the formula, we'd get plots of all variables against all others: why isn't that sensible here? What would happen if we used the formula Mobility ∼ Commute + Latitude + Longitude?*

The MLE of σ^2, $\hat{\sigma}^2$, is by

$$\hat{\sigma}^2 = \frac{1}{n}(\mathbf{y} - \mathbf{x}\widehat{\beta})^T(\mathbf{y} - \mathbf{x}\widehat{\beta}) \tag{4.38}$$

This estimator is negatively biased. Assuming that $\mathbf{x}$ is full rank, then

$$\mathbb{E}\left[\hat{\sigma}^2\right] = \frac{n-p-1}{n}\sigma^2,$$

and has the sampling distribution

$$\frac{n\hat{\sigma}^2}{\sigma^2} \sim \chi^2_{n-p-1} \tag{4.39}$$

$\hat{\sigma}^2 \frac{n}{n-p-1}$ is an unbiased estimator of σ^2.

4.4.1 $z-$ and $t-$ Tests for Single Coefficients

Denote by $\text{se}\left(\hat{\beta}_i\right)$ the true standard error of the estimator $\hat{\beta}_i$. From the general theory about the variance of $\widehat{\beta}$,

$$\text{se}\left(\hat{\beta}_i\right) = \sqrt{\sigma^2(\mathbf{x}^T\mathbf{x})^{-1}_{i+1,i+1}} \tag{4.40}$$

(We recall that by convention the first column of $\mathbf{x}$ corresponds to the intercept parameter). Further, from the Gaussian distribution of $\widehat{\beta}$,

$$\frac{\hat{\beta}_i - \beta_i}{\text{se}\left(\hat{\beta}_i\right)} \sim N(0,1) \tag{4.41}$$

If we know σ^2, so that we can compute $\text{se}\left(\hat{\beta}_i\right)$, we can use this to either test hypotheses about the exact value of β_i, or to form confidence intervals. Specifically, a $1-\alpha$ CI would be

$$\hat{\beta}_i \pm z(\alpha/2)\text{se}\left(\hat{\beta}_i\right) \tag{4.42}$$

with z_p being the p^{th} quantile of the standard Gaussian distribution.

If we use the unbiased estimate of σ^2, $\hat{\sigma}^2\frac{n}{n-p-1}$ (when we assume that $\mathbf{x}$ is full rank), to obtain an estimate $\hat{\text{se}}\left(\hat{\beta}_i\right)$, we find rather

$$\frac{\hat{\beta}_i - \beta_i}{\hat{\text{se}}\left(\hat{\beta}_i\right)} \sim t_{n-p-1}. \tag{4.43}$$

It follows that

$$\hat{\beta}_i \pm t_{n-p-1}(\alpha/2)\hat{\text{se}}\left(\hat{\beta}_i\right) \tag{4.44}$$

is a $1-\alpha$ confidence interval for β_i.

As $n \to \infty$, this becomes

$$\hat{\beta}_i \pm z(\alpha/2)\hat{\sigma}\sqrt{(\mathbf{x}^T\mathbf{x})^{-1}_{i+1,i+1}} \tag{4.45}$$

which is often a quite practical alternative to the t-based interval.

Wald t-tests in R When we run `summary` on the output of `lm`, part of what it delivers is a table containing estimated coefficients and standard errors, along with a t-statistic and a p-value for each one. The null hypothesis being tested here is different for each row of the table. The null hypothesis for β_i is that

$$Y = \beta_0 + \beta_1 X_1 + \ldots \beta_{i-1}X_{i-} + 0X_i + \beta_{i+1}X_i + \ldots + \beta_p X_p + \epsilon \tag{4.46}$$

with ϵ being mean-zero, constant-variance independent Gaussian noise. Whereas the alternative hypothesis is that

$$Y = \beta_0 + \beta_1 X_1 + \ldots \beta_{i-1}X_{i-} + \beta_i X_i + \beta_{i+1}X_i + \ldots + \beta_p X_p + \epsilon \tag{4.47}$$

with $\beta_i \neq 0$, and the same assumptions about ϵ. This matters because *whether the null hypothesis is true or not depends on what other variables are included in the model*. The optimal coefficient on X_i might be zero with one set of covariates and non-zero with another. The t test is, by its nature, incapable of saying whether X_i should be included in the model or not.

Example 1. *Suppose that the true model is*

$$Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \epsilon \tag{4.48}$$

with all the usual assumptions being met. We estimate instead the model

$$Y = \gamma_0 + \gamma_1 X_1 + \eta \tag{4.49}$$

We know, from our study of the simple linear model, that the (optimal or population) value of γ_1 is

$$\gamma_1 = \frac{\mathrm{Cov}\left[X_1, Y\right]}{\mathrm{Var}\left[X_1\right]} \tag{4.50}$$

Substituting in for Y,

$$\begin{aligned}
\gamma_1 &= \frac{\mathrm{Cov}\left[X_1, \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \epsilon\right]}{\mathrm{Var}\left[X_1\right]} && (4.51)\\
&= \frac{\mathrm{Cov}\left[X_1, \beta_0\right] + \mathrm{Cov}\left[X_1, \beta_1 X_1\right] + \mathrm{Cov}\left[X_1, \beta_2 X_2\right] + \mathrm{Cov}\left[X_1, \epsilon\right]}{\mathrm{Var}\left[X_1\right]} && (4.52)\\
&= \frac{0 + \beta_1 \mathrm{Cov}\left[X_1, X_1\right] + \beta_2 \mathrm{Cov}\left[X_1, X_2\right] + 0}{\mathrm{Var}\left[X_1\right]} && (4.53)\\
&= \beta_1 + \beta_2 \frac{\mathrm{Cov}\left[X_1, X_2\right]}{\mathrm{Var}\left[X_1\right]} && (4.54)
\end{aligned}$$

Thus, even if $\beta_1 = 0$, we can easily have $\gamma_1 \neq 0$, and vice versa.

4.4.2 F-Tests for utility of regressors

If we want to test whether a group of multiple coefficients are all simultaneously zero, the traditional approach is a variance ratio or F test. The null hypothesis is that

$$Y = \beta_0 + \beta_1 X_1 + \ldots + \beta_q X_q + 0 X_{q+1} + \ldots + 0 X_p + \epsilon \tag{4.55}$$

while the alternative is

$$Y = \beta_0 + \beta_1 X_1 + \ldots + \beta_q X_q + \beta_{q+1} X_{q+1} + \ldots + \beta_p X_p + \epsilon \tag{4.56}$$

with at least one of the coefficients $\beta_{q+1}, \ldots \beta_p \neq 0$. The null hypothesis, then, is that in a linear model which includes all the predictors $X_1, \ldots X_p$, the optimal coefficients for the last $p - q$ variables are all zero.

For both models, we get an estimate of σ^2, say $\hat{\sigma}^2_{null}$ for the null model (with coefficients fixed at zero) and $\hat{\sigma}^2_{full}$ for the full model. Because the null model is a special case of the full model, and we estimate parameters in each case by minimizing the MSE, $\hat{\sigma}^2_{null} \geq \hat{\sigma}^2_{full}$.

We have

$$\frac{n\hat{\sigma}^2_{full}}{\sigma^2} \sim \chi^2_{n-p-1} \tag{4.57}$$

while, under the null hypothesis,

$$\frac{n(\hat{\sigma}^2_{null} - \hat{\sigma}^2_{full})}{\sigma^2} \sim \chi^2_{p-q} \tag{4.58}$$

and so (again under the null hypothesis)

$$\frac{(\hat{\sigma}^2_{null} - \hat{\sigma}^2_{full})/(p-q)}{\hat{\sigma}^2_{full}/(n-p-1)} \sim F_{p-q,n-p-1} \tag{4.59}$$

We therefore reject the null hypothesis when the test statistic

$$F = \frac{(\hat{\sigma}^2_{null} - \hat{\sigma}^2_{full})/(p-q)}{\hat{\sigma}^2_{full}/(n-p-1)} \tag{4.60}$$

is too large compared to the $F_{p-q,n-p-1}$ distribution. This is why this is called an F test for this set of regression coefficients. If we're not testing all the coefficients at once, this is a **partial** F test.

The proper interpretation of this test is "Does letting the slopes for $X_{q+1}, \ldots X_p$ be non-zero reduce the MSE more than we would expect just by noise?" As n grows, increasingly small improvements will become clearly detectable as not-noise, so increasingly small but non-zero sets of coefficients will be detected as significant by the F test.

An obvious special case is the hypothesis that all the coefficients are zero. That is, the null hypothesis is

$$Y = \beta_0 + 0 X_1 + \ldots + 0 X_p + \epsilon \tag{4.61}$$

with the alternative being the full model

$$Y = \beta_0 + \beta_1 X_1 + \ldots + \beta_p X_p + \epsilon \tag{4.62}$$

The estimate of σ^2 under the null is the sample variance of Y, s_Y^2, so the test statistic becomes

$$\frac{(s_Y^2 - \hat{\sigma}^2_{full})/p}{\hat{\sigma}^2_{full}/(n-p-1)} \tag{4.63}$$

whose distribution under the null is $F_{p,n-p-1}$.

This **full** F test is often called a test of the significance of the whole regression. We are testing whether, if Y is linearly regressed on $X_1, \ldots X_p$ and only on those variables, the reduction in the MSE from actually estimating slopes over just using a flat regression surface is bigger than we'd expect from pure noise.

Variance Ratio Tests in R This is most easily done through the `anova` function. We fit the null model and the full model, both with `lm`, and then pass them to the `anova` function:

```
mobility <- read.csv("~/Dropbox/Cours Polytechnique/regression XHEC regression/data/mobility.csv")
mob.null <- lm(Mobility ~ Commute, data = mobility)
mob.full <- lm(Mobility ~ Commute + Latitude + Longitude, data = mobility)
anova(mob.null, mob.full)
## Analysis of Variance Table
##
## Model 1: Mobility ~ Commute
## Model 2: Mobility ~ Commute + Latitude + Longitude
##   Res.Df    RSS Df Sum of Sq      F   Pr(>F)
## 1    727 1.3143
## 2    725 1.2952  2  0.019111 5.3491 0.004942 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

The second row tells us that the full model has two more parameters than the null, that $n(\hat{\sigma}^2_{null} - \hat{\sigma}^2_{full}) = 0.0191114$, and then what the variance ratio or F statistic and the corresponding p-value are. Here, we learn that the decrease in the root-MSE which comes from adding latitude and longitude as predictors, while very small (0.51 percentage points) is large enough that it is unlikely to have arisen by capitalizing on noise.

Variable Deletion via F Tests We can use F tests to perform variable deletion: pick your least favorite set of predictors, test whether all of their βs are zero, and, if so, delete them from the model (and re-estimate).

Concretely, this test controls the probability of rejecting the null when the null is true — it guarantees that if $\beta_q = \mathbf{0}$, we have a low probability of rejecting that null hypothesis. For deletion to be reliable, however, we'd want a low probability of *missing* variables with non-zero coefficients, i.e., a low probability

of retaining the null hypothesis when it is wrong, or equivalently high power to detect departures from the null. Power cannot be read off from the p-value, and grows with the magnitude of the departure from the null. One way to get at this is to complement the hypothesis test with a confidence set for the coefficients in question. We can reasonably ignore variables whose coefficients are *precisely* estimated to be close to zero.

Remark 3. *a group of variables might show up as significant in a partial F test, even though none of them was individually significant on a t test in the full model. Also, if we delete variables in stages, we can have a situation where at each stage the increase in MSE is insignificant, but the difference between the full model and the final model is highly significant.*

Likelihood Ratio Tests There is an alternative to the F test based on the likelihood ratio. The log-likelihood of the model, at the maximum likelihood estimate, is

$$-\frac{n}{2}(1+\log 2\pi) - \frac{n}{2}\log \hat{\sigma}^2 \tag{4.64}$$

Hence the difference in log-likelihoods between the full model, with all p slopes estimated, and the null model, with only q slopes estimated and the other $p-q$ fixed, is

$$\Lambda = -\frac{n}{2}\log \hat{\sigma}^2_{full} + \frac{n}{2}\log \hat{\sigma}^2_{null} = \frac{n}{2}\log \frac{\hat{\sigma}^2_{null}}{\hat{\sigma}^2_{full}} \tag{4.65}$$

This is the log of the ratio of likelihoods.

Under the null hypothesis, as $n \to \infty$, we have,

$$2\Lambda \sim \chi^2_{p-q} \tag{4.66}$$

One advantage of likelihood ratio tests is that exactly the same procedure can be used to test the hypothesis that $\beta_q = \mathbf{0}$ and to test $\beta_q = \beta_q^*$, for any other particular vector of parameters. For that matter, it can be used to test $\mathbf{c}\beta = \mathbf{r}$, where $\mathbf{c}$ is any non-random $q \times (p+1)$ matrix, and $\mathbf{r}$ is any non-random $q \times 1$ vector. Thus, for example, it can be used to test the hypothesis that two slopes are *equal*, or that all slopes are equal, etc...

For linear-Gaussian models, both the likelihood ratio and the F statistic are functions of the ratio $\hat{\sigma}^2_{null}/\hat{\sigma}^2_{full}$. For fixed p and q, as $n \to \infty$, the two tests deliver the same p-values when $\hat{\sigma}^2_{null}/\hat{\sigma}^2_{full}$ is the same. At finite n, they are somewhat different, with the F test usually giving a somewhat higher p value than the χ^2 test, particularly if p is close to n. Which test is more *accurate* is another question. The likelihood ratio test can actually work for large n when the model is mis-specified, in the sense of telling us which wrong model is closer to the truth, while the F test's refinements over the χ^2 require all the modeling assumptions being right.

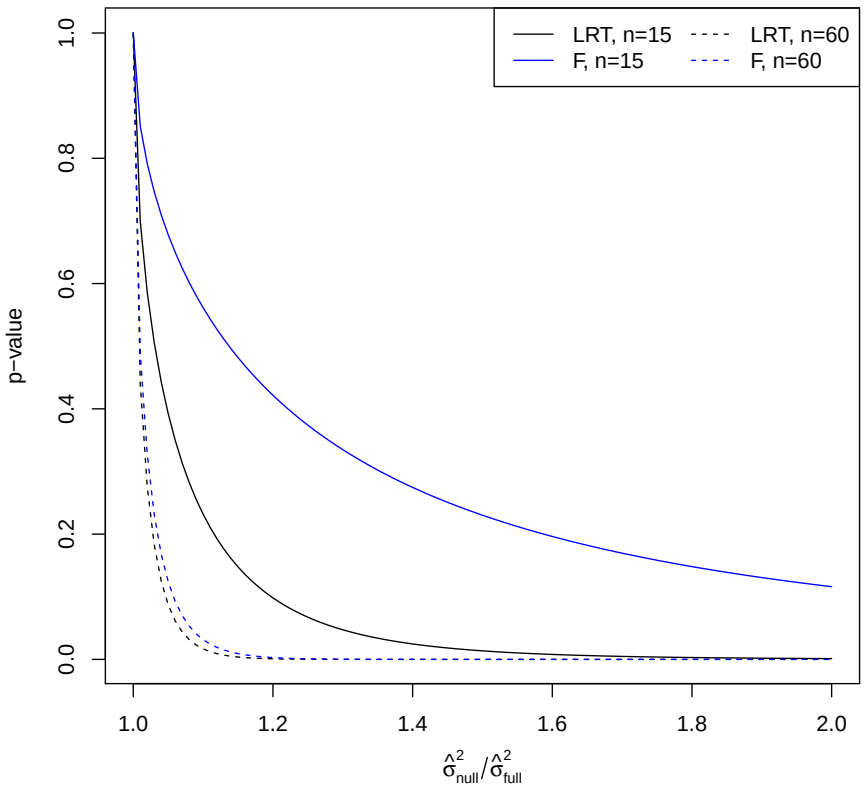


FIGURE 4.2: *Difference in p-values obtained from using a likelihood ratio test (black) and an F test (blue) on the same data, with $p = 10$, $q = 9$, and n either 15 (solid) or 60 (dotted). In general, the difference between the two tests goes to zero as $n - p$ grows. (See source file for code.)*

4.4.3 Confidence Sets for Multiple Coefficients

Suppose we want to do inference on two coefficients, say β_i and β_j, at once. That means we need to come up with a two-dimensional confidence region $C(\alpha)$, where we can say that $\mathbb{P}\left((\beta_i, \beta_j) \in C(\alpha)\right) = 1 - \alpha$. This would involve the same sort of trilemma as confidence intervals for single coefficients. That is, one of three things must be true:

1. Both β_i and β_j are in $C(\alpha)$; or
2. We got data which was very ($\leq \alpha$) improbable under all possible values of the parameters; or
3. Our model is wrong.

If we trust our model, then, we can indeed be confident that both β_i and β_j are simultaneously in $C(\alpha)$.

Clearly, nothing depends on wanting to do inference on just two coefficients at once; we could consider any subset of them we like, up to all $p+1$ of them.

With one parameter, intervals are the most natural confidence sets to work with. With more than one parameter, we have choices to make about the *shape* of the confidence set. The two easiest ones to work with are rectangular boxes, and ellipsoids.

4.4.4 Confidence Boxes or Rectangles

The natural thing to want to do is to take a confidence interval for each coefficient and put them together into a confidence box or rectangle. For instance, using the t-distribution CI for β_i and β_j, the box would be

$$(\hat{\beta}_i \pm t_{n-p-1}(\alpha/2)\hat{\text{se}}\left(\hat{\beta}_i\right)) \times (\hat{\beta}_j \pm t_{n-p-1}(\alpha/2)\hat{\text{se}}\left(\hat{\beta}_j\right)) \tag{4.67}$$

(And similarly for three or more parameters.) This is, however, not quite right as I've written it. The problem is that while each interval covers its true coefficient with high probability, both intervals *simultaneously* cover the pair of parameters is a different story. Let me abbreviate the interval for β_i as $C_i(\alpha)$, likewise the interval for β_j is $C_j(\alpha)$. We have

$$\mathbb{P}\left(\beta_i \in C_i(\alpha)\right) = 1 - \alpha \ , \ \mathbb{P}\left(\beta_j \in C_j(\alpha)\right) = 1 - \alpha \tag{4.68}$$

but from this it does not follow that

$$\mathbb{P}\left(\beta_i \in C_i(\alpha), \beta_j \in C_j(\alpha)\right) = 1 - \alpha \tag{4.69}$$

To see this, let's consider the complementary event: it's

$$\beta_i \notin C_i(\alpha) \vee \beta_j \notin C_j(\alpha) \tag{4.70}$$

writing $\vee$ for logical-or[1] By basic probability,

$$\mathbb{P}\left(\beta_i \notin C_i(\alpha) \vee \beta_j \notin C_j(\alpha)\right) = \mathbb{P}\left(\beta_i \notin C_i(\alpha)\right) + \mathbb{P}\left(\beta_j \notin C_j(\alpha)\right) - \mathbb{P}\left(\beta_i \notin C_i(\alpha), \beta_j \notin C_j(\alpha)\right) \tag{4.71}$$

Since C_i and C_j are $1-\alpha$-confidence sets,

$$\mathbb{P}\left(\beta_i \notin C_i(\alpha) \vee \beta_j \notin C_j(\alpha)\right) = 2\alpha - \mathbb{P}\left(\beta_i \notin C_i(\alpha), \beta_j \notin C_j(\alpha)\right) \leq 2\alpha \tag{4.72}$$

So $C_i(\alpha) \times C_j(\alpha)$ isn't itself a $1-\alpha$ confidence set; its real confidence level could be as little as $1-2\alpha$. If we had been looking at q coefficients at once, the confidence level might have been as low as $1-q\alpha$.

This suggests, however, a very simple, if sometimes over-cautious, way of building a confidence box. If we want the final box to have a $1-\alpha$ confidence level, and we're dealing with q coefficients at once, we calculate $1-\alpha/q$ confidence levels for each coefficient, say $C_i(\alpha/q)$, and then set

$$C(\alpha) = C_1(\alpha/q) \times C_2(\alpha/q) \times \ldots \times C_q(\alpha/q) \tag{4.73}$$

By our reasoning above, this final $C(\alpha)$ will cover all q parameters at once with probability at least $1-\alpha$.

This trick of building a $1-\alpha$ confidence box for q parameters at once from q $1-\alpha/q$ confidence intervals is completely generic and is known as Bonferroni correction; it works for any parameters of any statistical model and does not require independence of the observations. Bonferroni correction is also often used for hypothesis testing: if we test q distinct hypotheses, and we want to have the probability of making *no* false rejections be α, we can achieve that by having each test be of size α/q.

4.4.5 Confidence Balls or Ellipsoids

An alternative to confidence boxes is to try to make confidence *balls*. To see how this could work, suppose first that $\hat{\beta}_i$ and $\hat{\beta}_j$ were uncorrelated. Since

$$\frac{\hat{\beta}_i - \beta_i}{\mathrm{se}\left(\hat{\beta}_i\right)} \sim N(0,1) \tag{4.74}$$

(and likewise for β_j), we would have[2]

$$\left(\frac{\hat{\beta}_i - \beta_i}{\mathrm{se}\left(\hat{\beta}_i\right)}\right)^2 + \left(\frac{\hat{\beta}_j - \beta_j}{\mathrm{se}\left(\hat{\beta}_j\right)}\right)^2 \sim \chi^2_2 \tag{4.75}$$

Therefore, a simultaneous $1-\alpha$ confidence region for (β_i, β_j) would be the region where

$$\left(\frac{\hat{\beta}_i - \beta_i}{\mathrm{se}\left(\hat{\beta}_i\right)}\right)^2 + \left(\frac{\hat{\beta}_j - \beta_j}{\mathrm{se}\left(\hat{\beta}_j\right)}\right)^2 \leq \chi^2_2(1-\alpha) \tag{4.76}$$

[1] That is, $A \vee B$ means in ordinary English "A is true or B is true or both are true".

[2] Because when $Z_1, \ldots Z_d$ are independent $N(0,1)$ variables, $\sum_i Z_i^2 \sim \chi^2_d$.

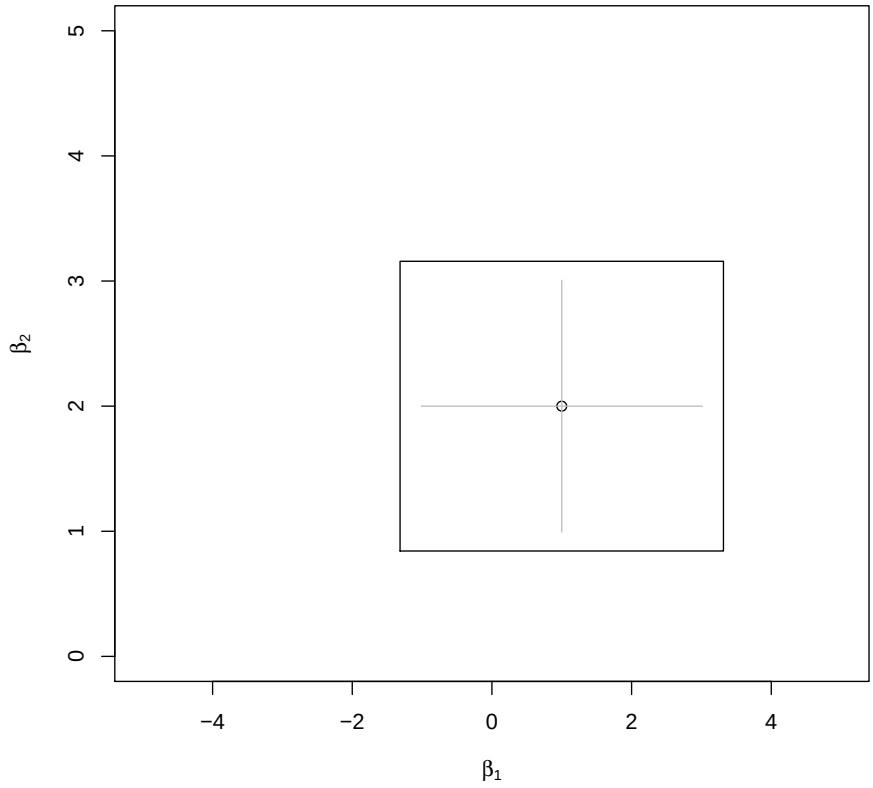


FIGURE 4.3: *Grey lines: 95% confidence intervals for two coefficients, based on inverting t tests, and so centered at the point estimate (dot). Black box: a 95% confidence rectangle for both coefficients simultaneously. Notice that the grey lines do not touch the sides of the rectangle; the latter correspond to 97.5% CIs for each coefficient. If we did draw the rectangle corresponding to the grey lines, its actual confidence level could be as low as 90%. (See source file for code.)*

A little geometry shows that this region is an ellipse, its axes parallel to the coordinate axis with the length from end to end along one axis being $2\text{se}\left(\hat{\beta}_i\right)\chi^2_2(1-\alpha)$, and its length along the other axis being $2\text{se}\left(\hat{\beta}_j\right)\chi^2_2(1-\alpha)$.

If we had q different uncorrelated coefficients, the confidence region would be the set $(\beta_1, \beta_2, \ldots \beta_q)$ where

$$\sum_{i=1}^{q}\left(\frac{\hat{\beta}_i - \beta_i}{\text{se}\left(\hat{\beta}_i\right)}\right)^2 \leq \chi^2_q(1-\alpha) \tag{4.77}$$

When $q > 2$, we call this region an "ellipsoid" rather than an "ellipse", but it's the same idea.

Usually, of course, the different coefficient estimates are correlated with each other, so we need to do something a bit different. If we write β_q for the vector of coefficients we're interested in, and $\mathbf{\Sigma}_q$ for its variance-covariance matrix, then the confidence region is the set of all β_q where

$$(\widehat{\beta}_q - \beta_q)^T \mathbf{\Sigma}_q^{-1}(\widehat{\beta}_q - \beta_q) \leq \chi^2_q(1-\alpha) \tag{4.78}$$

This, too, is an ellipsoid, only now the axes point in the directions given by the eigenvectors of $\mathbf{\Sigma}_q$, and the length along each axis is proportional to the square root of the corresponding eigenvalue. (See §**??** for a derivation.)

Since $\mathbf{\Sigma}_q$ is a $q \times q$ sub-matrix of $\sigma^2(\mathbf{x}^T\mathbf{x})^{-1}$, we can't actually use this. We can, however, use the appropriate sub-matrix of $\hat{\sigma}^2(\mathbf{x}^T\mathbf{x})^{-1}$ as an approximation, which becomes exact as $n \to \infty$. Similarly, if we use the unbiased estimate of σ^2, we replace $\chi^2_q(1-\alpha)$ with $F_{q,n-p-1}(1-\alpha)$.

Proof. We note first that $\mathbf{\Sigma}_q$ is a square, symmetric, positive-definite matrix. Therefore it can be written as follows:

$$\mathbf{\Sigma}_q = \mathbf{V}\mathbf{U}\mathbf{V}^T \tag{4.79}$$

where $\mathbf{U}$ is the diagonal matrix of eigenvalues, and $\mathbf{V}$ is the matrix whose columns are the eigenvectors; $\mathbf{V}^T$ is its transpose, and $\mathbf{V}^T\mathbf{V} = \mathbf{I}$. If we define $\mathbf{\Sigma}_q^{1/2} = \mathbf{V}\mathbf{U}^{1/2}$, where $\mathbf{U}^{1/2}$ is the diagonal matrix with the square roots of the eigenvalues, then

$$\begin{aligned}
\text{Var}\left[\mathbf{\Sigma}_q^{-1/2}(\widehat{\beta}_q - \beta_q)\right] &= \mathbf{\Sigma}_q^{-1/2}\text{Var}\left[\widehat{\beta}_q - \beta_q)\right](\mathbf{\Sigma}_q^{-1/2})^T && (4.80)\\
&= \mathbf{U}^{-1/2}\mathbf{V}^{-1}\mathbf{V}\mathbf{U}\mathbf{V}^T\mathbf{V}\mathbf{U}^{-1/2} && (4.81)\\
&= \mathbf{U}^{-1/2}\mathbf{U}\mathbf{U}^{-1/2} && (4.82)\\
&= \mathbf{I} && (4.83)
\end{aligned}$$

where the last step works because $\mathbf{U}$ and $\mathbf{U}^{-1/2}$ are both diagonal matrices. In other words, while the coordinates of $\widehat{\beta}_q - \beta_q$ have unequal variances and

are correlated with each other, $\mathbf{\Sigma}_q^{-1/2}(\widehat{\beta}_q - \beta_q)$ is a random vector where each coordinate has variance 1 and is uncorrelated with the others. Since the initial vector was Gaussian, this too is Gaussian, hence

$$\mathbf{\Sigma}_q^{-1/2}(\widehat{\beta}_q - \beta_q) \sim N(\mathbf{0}, \mathbf{I}) \tag{4.84}$$

Therefore

$$\left(\mathbf{\Sigma}_q^{-1/2}(\widehat{\beta}_q - \beta_q)\right)^T \mathbf{\Sigma}_q^{-1/2}(\widehat{\beta}_q - \beta_q) \sim \chi^2_q \tag{4.85}$$

since it's a sum of q squared, independent $N(0,1)$ variables.

On the other hand,

$$\begin{aligned}
&\left(\mathbf{\Sigma}_q^{-1/2}(\widehat{\beta}_q - \beta_q)\right)^T \left(\mathbf{\Sigma}_q^{-1/2}(\widehat{\beta}_q - \beta_q)\right) && (4.86)\\
&= (\widehat{\beta}_q - \beta_q)^T \left(\mathbf{\Sigma}_q^{-1/2}\right)^T \mathbf{\Sigma}_q^{-1/2}(\widehat{\beta}_q - \beta_q) && \\
&= (\widehat{\beta}_q - \beta_q)^T \mathbf{V}\mathbf{U}^{-1/2}\mathbf{U}^{-1/2}\mathbf{V}^{-1}(\widehat{\beta}_q - \beta_q) && (4.87)\\
&= (\widehat{\beta}_q - \beta_q)^T \mathbf{V}\mathbf{U}^{-1}\mathbf{V}^{-1}(\widehat{\beta}_q - \beta_q) && (4.88)\\
&= (\widehat{\beta}_q - \beta_q)^T \mathbf{\Sigma}_q^{-1}(\widehat{\beta}_q - \beta_q) && (4.89)
\end{aligned}$$

Combining Eqs. 4.85 and 4.89,

$$(\widehat{\beta}_q - \beta_q)^T \mathbf{\Sigma}_q^{-1}(\widehat{\beta}_q - \beta_q) \sim \chi^2_q \tag{4.90}$$

as was to be shown.

Confidence Ellipsoids in R. The package `ellipse` (**?**) contains functions for plotting 2D confidence ellipses. The main function is also called `ellipse`, which happens to have a specialized method for `lm` models. The usage is

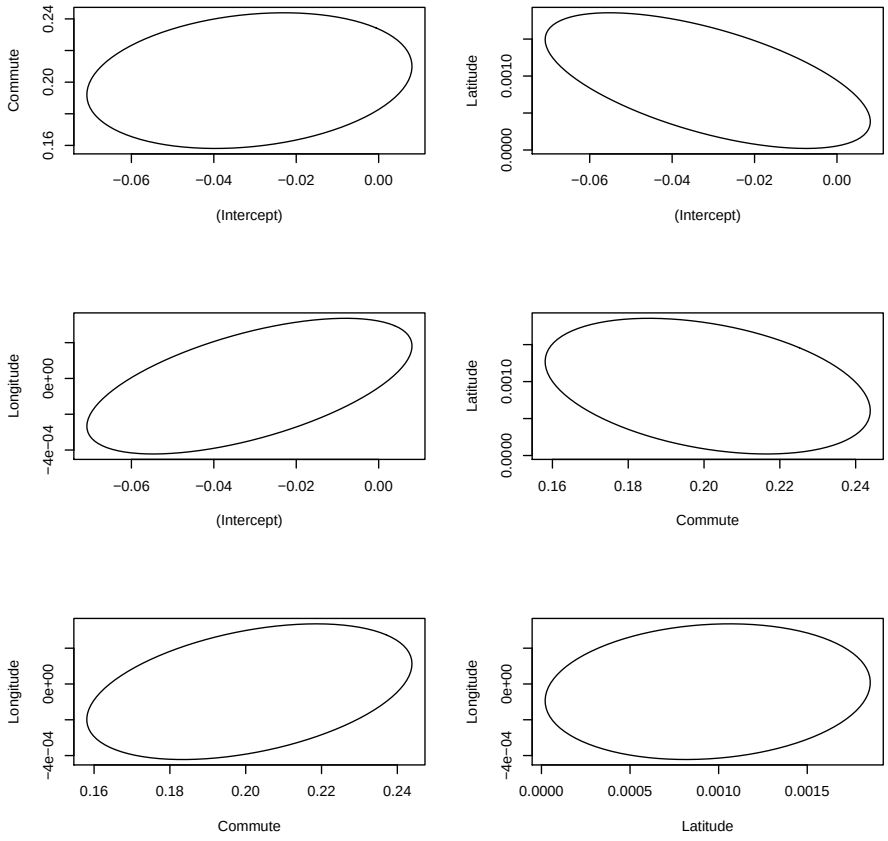
```
my.model <- lm(y ~ x1 + x2 + x3)
plot(ellipse(my.model, which = c(1, 2), level = 0.95))
```

Here `which` is the vector of coefficient indices (it can only be of length 2) and `level` is the confidence level. Notice that what `ellipse` actually returns is a two-column array of coordinates, which can be plotted, or passed along to other graphics functions (like `points` or `lines`). See Figure 4.4.

Three-dimensional confidence ellipsoids can be made with the `rgl` library (**?**). While confidence ellipsoids exist in any number of dimensions, they can't really be visualized when $q > 3$.

4.5 Diagnostics for Multiple Linear Regression

We made several assumptions on our linear model to perform detailed statistical inference. We need to check these modeling assumptions. which means we need diagnostics.




```
library(ellipse)
par(mfrow = c(3, 2))
plot(ellipse(mob.full, which = c(1, 2), level = 1 - 0.05/6), type = "l")
plot(ellipse(mob.full, which = c(1, 3), level = 1 - 0.05/6), type = "l")
plot(ellipse(mob.full, which = c(1, 4), level = 1 - 0.05/6), type = "l")
plot(ellipse(mob.full, which = c(2, 3), level = 1 - 0.05/6), type = "l")
plot(ellipse(mob.full, which = c(2, 4), level = 1 - 0.05/6), type = "l")
plot(ellipse(mob.full, which = c(3, 4), level = 1 - 0.05/6), type = "l")
```

FIGURE 4.4: *Confidence ellipses for every pair of coefficients in the model where economic mobility is regressed on the prevalence of short commutes, latitude and longitude. (Remember the intercept is the first coefficient.) Why do I use this odd-looking confidence level?*

Visual diagnostics. All of the plots we learned how to do for simple linear regression remain valuable:

1. *Plot the residuals against the predictors.* This now means p distinct plots, of course. Each of them should show a flat scatter of points around 0 (because $\mathbb{E}[\epsilon|X_i] = 0$), of roughly constant width (because $\mathrm{Var}(\epsilon|X_i) = \sigma^2$). Curvature or steps to this plot is a sign of potential nonlinearity, or of an omitted variable. Changing width is a potential sign of non-constant variance.

2. *Plot the squared residuals against the predictors.* Each of these p plots should show a flat scatter of points around $\hat{\sigma}^2$.

3. *Plot the residuals against the fitted values.* This is an extra plot, redundant when we only have one predictor (because the fitted values were linear in the predictor).

4. *Plot the squared residuals against the fitted values.*

5. *Plot the residuals against coordinates.* If observations are dated, time-stamped, or spatially located, plot the residuals as functions of time, or make a map. If there is a meaningful order to the observations, plot residuals from successive observations against each other. Because the ϵ_i are uncorrelated, all of these plots should show a lack of structure.

6. *Plot the residuals' distribution against a Gaussian.*

Out-of-sample predictions, with either random or deliberately selected testing sets, also remain valuable.

Linear dependence. A linear dependence between two (or more) columns of the $\mathbf{x}$ matrix is called **collinearity** or **linear dependence**, and it keeps us from finding a solution by least squares. (In fact, collinearity at the population level makes the coefficients non identifiable). Collinearity between a pair of variables will show up in a pairs plot as an exact straight line. Collinearity among more than two variables will not. For instance, if $X_3 = (X_1 + X_2)/2$, we can't include all three variables in a regression, but we'd not see that from any of the pairs.

Computationally, collinearity will show up in the form of the determinant of $\mathbf{x}^T\mathbf{x}$ being zero. Equivalently, the smallest eigenvalue of $\mathbf{x}^T\mathbf{x}$ will be zero. If `lm` is given a collinear set of predictor variables, it will sometimes give an error messages, but more often it will decide not to estimate one of the collinear variables, and return an `NA` for the offending coefficient.

Chapter 5

Outliers and Influential Point

An **outlier** is a data point which is very far, somehow, from the rest of the data. They are often worrisome, but not always a problem. When we are doing regression modeling, in fact, we don't really care about whether some data point is far from the rest of the data, but whether it breaks a pattern the rest of the data seems to follow. Here, we'll first try to build some intuition for when outliers cause trouble in linear regression models. Then we'll look at some ways of quantifying how much influence particular data points have on the model; consider the strategy of pretending that inconvenient data doesn't exist; and take a brief look at the **robust regression** strategy, of replacing least squares estimates with others which are less easily influenced.

5.1 Outliers Are Data Points Which Break a Pattern

Consider Figure 5.1. The points marked in red and blue are clearly not like the main cloud of the data points, even though their x and y coordinates are quite typical of the data as a whole: the x coordinates of those points aren't related to the y coordinates in the right way, they break a pattern. On the other hand, the point marked in green, while its coordinates are very weird on both axes, does not break that pattern — it was positioned to fall right on the regression line.

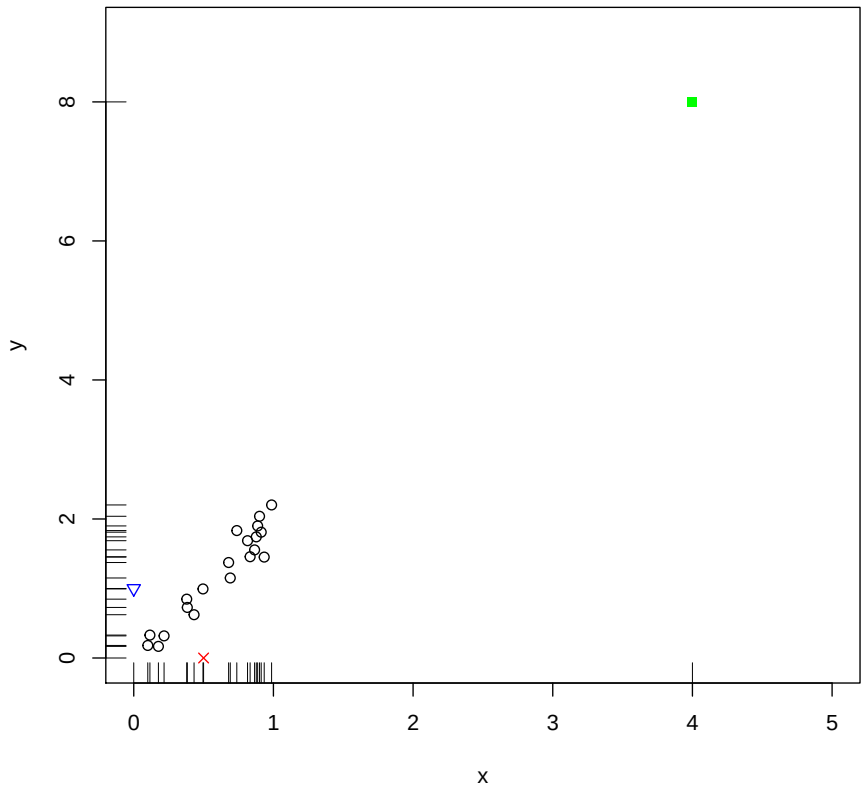


FIGURE 5.1: *Points marked with a red* $\times$ *and a blue triangle are outliers for the regression line through the main cloud of points, even though their* x *and* y *coordinates are quite typical of the marginal distributions (see rug plots along axes). The point marked by the green square, while an outlier along both axes, falls right along the regression line. (See the source file online for the figure-making code.)*

	(Intercept)	x
black only	-0.06520	2.07
black+blue	-0.15000	2.13
black+red	0.16200	1.77
black+green	-0.03350	2.02
all points	-0.00147	1.98

TABLE 5.1: *Estimates of the simple regression line from the black points in Figure 5.1, plus re-estimates adding in various outliers.*

What affect do these different outliers have on a simple linear model here? Table 5.1 shows the estimates we get from using just the black points, from adding only one of the three outlying points to the black points, and from using all the points. As promised, adding the red or blue points shifts the line, while adding the green point changes hardly anything at all.

If we are worried that outliers might be messing up our model, we would like to quantify how much the estimates change if we add or remove individual data points. Fortunately, we can quantify this using only quantities we estimated on the complete data, especially the hat matrix.

5.1.1 Examples with Simple Linear Regression

To further build intuition, let's think about what happens with simple linear regression for a moment; that is, our model is

$$Y = \beta_0 + \beta_1 X + \epsilon \tag{5.1}$$

with a single real-valued predictor variable X. When we estimate the coefficients by least squares, we know that

$$\hat{\beta}_0 = \overline{y} - \hat{\beta}_1 \overline{x} \tag{5.2}$$

Let us turn this around. The fitted value at $X = \overline{x}$ is

$$\hat{\beta}_0 + \hat{\beta}_1 \overline{x} = \overline{y} \tag{5.3}$$

Suppose we had a data point, say the i^{th} point, where $X = \overline{x}$. Then the actual value of y_i almost wouldn't matter for the fitted value there — the regression line *has* to go through $\overline{y}$ at $\overline{x}$, never mind whether y_i there is close to $\overline{y}$ or far away. If $x_i = \overline{x}$, we say that y_i has little *leverage* over $\hat{m}_i$, or little *influence* on $\hat{m}_i$. It has *some* influence, because y_i is part of what we average to get $\overline{y}$, but that's not a lot of influence.

Again, with simple linear regression, we know that

$$\hat{\beta}_1 = \frac{c_{XY}}{s^2_X} \tag{5.4}$$

the ratio between the sample covariance of X and Y and the sample variance of X. How does y_i show up in this? It's

$$\hat{\beta}_1 = \frac{n^{-1}\sum_{i=1}^{n}(x_i - \overline{x})(y_i - \overline{y})}{s^2_X} \tag{5.5}$$

Notice that when $x_i = \overline{x}$, y_i doesn't actually matter at all to the slope. If x_i is far from $\overline{x}$, then $y_i - \overline{y}$ will contribute to the slope, and its contribution will get bigger (whether positive or negative) as $x_i - \overline{x}$ grows. y_i will also make a big contribution to the slope when $y_i - \overline{y}$ is big (unless, again, $x_i = \overline{x}$).

Let's write a general formula for the predicted value, at an arbitrary point $X = x$.

$$\begin{aligned}
\hat{m}(x) &= \hat{\beta}_0 + \hat{\beta}_1 x && (5.6)\\
&= \overline{y} - \hat{\beta}_1 \overline{x} + \hat{\beta}_1 x && (5.7)\\
&= \overline{y} + \hat{\beta}_1 (x - \overline{x}) && (5.8)\\
&= \overline{y} + \frac{1}{n}\frac{\sum_{i=1}^{n}(x_i - \overline{x})(y_i - \overline{y})}{s^2_X}(x - \overline{x}) && (5.9)
\end{aligned}$$

So, in words:

- The predicted value is always a weighted average of all the y_i.

- As x_i moves away from $\overline{x}$, y_i gets more weight (possibly a large negative weight). When $x_i = \overline{x}$, y_i only matters because it contributes to the global mean $\overline{y}$.
- The weights on all data points increase in magnitude when the point x where we're trying to predict is far from $\overline{x}$. If $x = \overline{x}$, only $\overline{y}$ matters.

All of this is still true of the fitted values at the original data points:

- If x_i is at $\overline{x}$, y_i only matters for the fit because it contributes to $\overline{y}$.
- As x_i moves away from $\overline{x}$, in either direction, it makes a bigger contribution to *all* the fitted values.

Why is this happening? We get the coefficient estimates by minimizing the mean squared error, and the MSE treats all data points equally:

$$\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{m}(x_i))^2 \tag{5.10}$$

But we're not just using any old function $\hat{m}(x)$; we're using a linear function. This has only two parameters, so we can't change the predicted value to match each data point — altering the parameters to bring $\hat{m}(x_i)$ closer to y_i might actually increase the error elsewhere. By minimizing the over-all MSE with a linear function, we get two constraints,

$$\overline{y} = \hat{\beta}_0 + \hat{\beta}_1 \overline{x} \tag{5.11}$$

and

$$\sum_i e_i(x_i - \overline{x}) = 0 \tag{5.12}$$

The first of these makes the regression line insensitive to y_i values when x_i is close to $\overline{x}$. The second makes the regression line *very* sensitive to residuals when $x_i - \overline{x}$ is big — when $x_i - \overline{x}$ is large, a big residual (e_i far from 0) is harder to balance out than if $x_i - \overline{x}$ were smaller.

So, let's sum this up.

- Least squares estimation tries to bring all the predicted values closer to y_i, but it can't match each data point at once, because the fitted values are all functions of the same coefficients.
- If x_i is close to $\overline{x}$, y_i makes little difference to the coefficients or fitted values — they're pinned down by needing to go through the mean of the data.
- As x_i moves away from $\overline{x}$, $y_i - \overline{y}$ makes a bigger and bigger impact on both the coefficients and on the fitted values.

If we worry that some point isn't falling on the same regression line as the others, we're really worrying that including it will throw off our estimate of the line. This is going to be a concern when x_i is far from $\overline{x}$, or when the combination of $x_i - \overline{x}$ and $y_i - \overline{y}$ makes that point has a disproportionate impact on the estimates. We should also be worried if the residual values are too big, but when asking what's "too big", we need to take into account the fact that the model will try harder to fit some points than others. A big residual at a point of high leverage is more of a red flag than an equal-sized residual at point with little influence.

All of this will carry over to multiple regression models, but with more algebra to keep track of the different dimensions.

5.2 Influence of Individual Data Points on Estimates

Recall that our least-squares coefficient estimator is

$$\widehat{\beta} = (\mathbf{x}^T\mathbf{x})^{-1}\mathbf{x}^T\mathbf{y} \tag{5.13}$$

from which we get our fitted values as

$$\widehat{\mathbf{m}} = \mathbf{x}\widehat{\beta} = \mathbf{x}(\mathbf{x}^T\mathbf{x})^{-1}\mathbf{x}^T\mathbf{y} = \mathbf{H}\mathbf{y} \tag{5.14}$$

with the hat matrix $\mathbf{H} \equiv \mathbf{x}(\mathbf{x}^T\mathbf{x})^{-1}\mathbf{x}^T$. This leads to a very natural sense in which one observation might be more or less influential than another:

$$\frac{\partial \hat{\beta}_k}{\partial y_i} = \left((\mathbf{x}^T\mathbf{x})^{-1}\mathbf{x}^T\right)_{ki} \tag{5.15}$$

and

$$\frac{\partial \hat{m}_k}{\partial y_i} = H_{ii} \tag{5.16}$$

If y_i were different, it would change the estimates for all the coefficients and for all the fitted values. The rate at which the k^{th} coefficient or fitted value changes is given by the ki^{th} entry in these matrices — matrices which, notice, are completely defined by the design matrix $\mathbf{x}$.

5.2.1 Leverage

H_{ii} is the influence of y_i on its own fitted value; it tells us how much of $\hat{m}_i$ is just y_i. This turns out to be a key quantity in looking for outliers, so we'll give it a special name, the **leverage**. It is sometimes also written h_i. Once again, the leverage of the i^{th} data point doesn't depend on y_i, only on the design matrix.

Because the general linear regression model doesn't assume anything about the distribution of the predictors, other than that they're not collinear, we can't say definitely that some values of the leverage break model assumptions, or even are very unlikely under the model assumptions. But we can say some things about the leverage.

Average leverages We showed in the homework that the trace of the hat matrix equals the number of coefficients we estimate:

$$\mathrm{tr}\,(\mathbf{H}) = p + 1 \tag{5.17}$$

But the trace of any matrix is the sum of its diagonal entries,

$$\mathrm{tr}\,(\mathbf{H}) = \sum_{i=1}^{n} H_{ii} \tag{5.18}$$

so the trace of the hat matrix is the sum of each point's leverage. The average leverage is therefore $\frac{p+1}{n}$. We don't expect every point to have exactly the same leverage, but if some points have much more than others, the regression function is going to be pulled towards fitting the high-leverage points, and the function will tend to ignore the low-leverage points.

Leverage vs. geometry Let's center all the predictor variables, i.e., subtract off the mean of each predictor variable. Call this new vector of predictor variables Z, with the $n \times p$ matrix $\mathbf{z}$. This will not change any of the slopes, and will fix the intercept to be $\overline{y}$. The fitted values then come from

$$\hat{m}_i = \overline{y} + \frac{1}{n}(\mathbf{x}_i - \overline{\mathbf{x}})\mathrm{Var}\,[X]^{-1}\,\mathbf{z}^T\mathbf{y} \tag{5.19}$$

This tells us that y_i will have a lot of leverage if $(\mathbf{x}_i - \overline{\mathbf{x}})\mathrm{Var}\,[X]^{-1}\,(\mathbf{x}_i - \overline{\mathbf{x}})^T$ is big[1]. If the data point falls exactly at the mean of the predictors, y_i matters only because it contributes to the over-all mean $\overline{y}$. If the data point moves away from the mean of the predictors, not all directions count equally. Remember the eigen-decomposition of $\mathrm{Var}\,[X]$:

$$\mathrm{Var}\,[X] = \mathbf{V}\mathbf{U}\mathbf{V}^T \tag{5.20}$$

where $\mathbf{V}$ is the matrix whose columns are the eigenvectors of $\mathrm{Var}\,[X]$, $\mathbf{V}^T = \mathbf{V}^{-1}$, and $\mathbf{U}$ is the diagonal matrix of the eigenvalues of $\mathrm{Var}\,[X]$. Each eigenvalue gives the variance of the predictors along the direction of the corresponding eigenvector. It follows that

$$\mathrm{Var}\,[X]^{-1} = \mathbf{V}\mathbf{U}^{-1}\mathbf{V} \tag{5.21}$$

So if the data point is far from the center of the predictors along a high-variance direction, that doesn't count as much as being equally far along a low-variance direction. Figure 5.2 shows a distribution for two predictor variables we're very familiar with, together with the two eigenvectors from the variance matrix, and the corresponding surface of leverages.

You may convince yourself that with one predictor variable, all of this collapses down to just $1/n + (x_i - \overline{x})^2/ns_X^2$. This leads to plots which may be easier to grasp (Figure 5.3).

[1]This sort of thing — take the difference between two vectors, multiply by an inverse variance matrix, and multiply by the difference vector again — is called a **Mahalanobis**

```
## Error in library(scatterplot3d):  there is no package called
'scatterplot3d'
## Error in eval(expr, envir, enclos):  could not find function
"scatterplot3d"
```

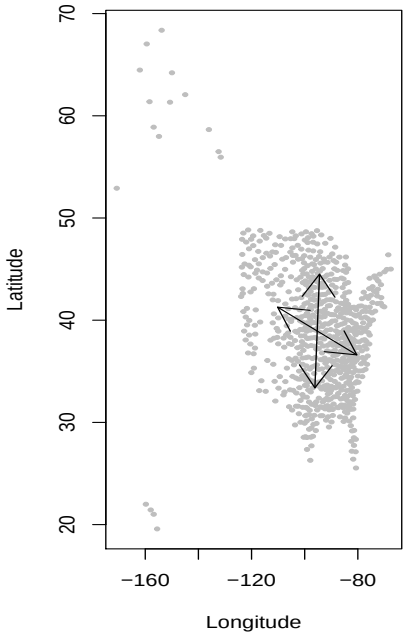
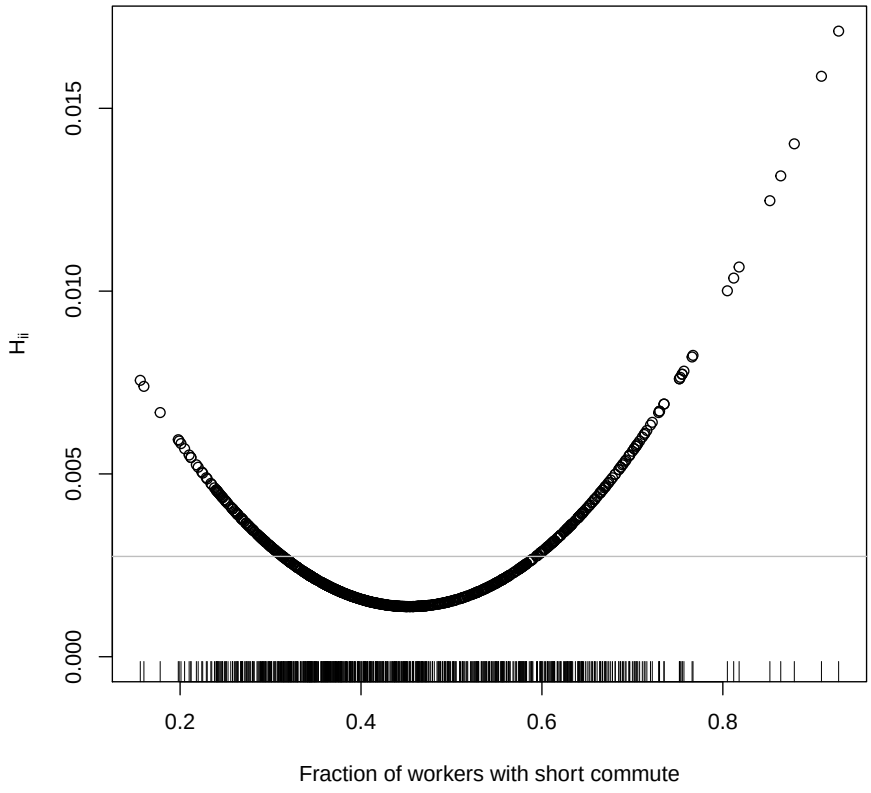


FIGURE 5.2: *Left: The geographic coordinates of the communities from the* `mobility` *data, along with their mean, and arrows marking the eigenvectors of the variance-covariance matrix (lengths scaled by the eigenvalues). Right: leverages for each point when regressing rates of economic mobility (or anything else) on latitude and longitude. See online for the code.*




```
H.mob.lm <- hatvalues(lm(Mobility ~ Commute, data = mobility))
plot(mobility$Commute, H.mob.lm, ylim = c(0, max(H.mob.lm)), xlab = "Fraction of workers with short commute"
    ylab = expression(H[ii]))
abline(h = 2/nrow(mobility), col = "grey")
rug(mobility$Commute, side = 1)
```

FIGURE 5.3: *Leverages (H_{ii}) for a simple regression of economic mobility (or anything else) against the fraction of workers with short commutes. The grey line marks the average we'd see if every point was exactly equally influential. Note how leverage increases automatically as* `Commute` *moves away from its mean in either direction. (See below for the* `hatvalues` *function.*

One curious feature of the leverage is, and of the hat matrix in general, is that it doesn't care *what* we are regressing on the predictor variables; it could be economic mobility or sightings of Bigfoot, and the same design matrix will give us the same hat matrix and leverages.

To sum up: The leverage of a data point just depends on the value of the predictors there; it increases as the point moves away from the mean of the predictors. It increases more if the difference is along low-variance coordinates, and less for differences along high-variance coordinates.

5.3 Studentized Residuals

We return once more to the hat matrix, the source of all knowledge.

$$\widehat{\mathbf{m}} = \mathbf{H}\mathbf{y} \tag{5.22}$$

The residuals, too, depend only on the hat matrix:

$$\mathbf{e} = \mathbf{y} - \widehat{\mathbf{m}} = (\mathbf{I} - \mathbf{H})\mathbf{y} \tag{5.23}$$

We know that the residuals vary randomly with the noise, so let's re-write this in terms of the noise.

$$\mathbf{e} = (\mathbf{I} - \mathbf{H})\epsilon \tag{5.24}$$

Since $\mathbb{E}[\epsilon] = \mathbf{0}$ and $\mathrm{Var}[\epsilon] = \sigma^2\mathbf{I}$, we have

$$\mathbb{E}[\mathbf{e}] = \mathbf{0} \tag{5.25}$$

and

$$\mathrm{Var}[\mathbf{e}] = \sigma^2(\mathbf{I} - \mathbf{H})(\mathbf{I} - \mathbf{H})^T = \sigma^2(\mathbf{I} - \mathbf{H}) \tag{5.26}$$

If we also assume that the noise is Gaussian, the residuals are Gaussian, with the stated mean and variance.

What does this imply for the residual at the i^{th} data point? It has expectation 0,

$$\mathbb{E}[e_i] = 0 \tag{5.27}$$

and it has a variance which depends on i through the hat matrix:

$$\mathrm{Var}[e_i] = \sigma^2(\mathbf{I} - \mathbf{H})_{ii} = \sigma^2(1 - H_{ii}) \tag{5.28}$$

In words: the bigger the leverage of i, the smaller the variance of the residual there. This is yet another sense in which points with high leverage are points which the model tries very hard to fit.

Previously, when we looked at the residuals, we expected them to all be of roughly the same magnitude. This rests on the leverages H_{ii} being all about the

distance. As we will see in a moment, it gives more attention to differences along coordinates where the variance is small, and less attention to differences along coordinates where the variance is high.

same size. If there are substantial variations in leverage across the data points, it's better to scale the residuals by their expected size.

The usual way to do this is through the **standardized** or **studentized residuals**

$$r_i \equiv \frac{e_i}{\hat{\sigma}\sqrt{1-H_{ii}}} \tag{5.29}$$

Why "studentized"? Because we're dividing by an estimate of the standard error, just like in "Student's" t-test for differences in means All of the residual plots we've done before can also be done with the studentized residuals. In particular, the studentized residuals should look flat, with constant variance, when plotted against the fitted values or the predictors.

5.4 Leave-One-Out

Suppose we left out the i^{th} data point altogether. How much would that change the model?

5.4.1 Fitted Values and Cross-Validated Residuals

Let's take the fitted values first. The hat matrix, $\mathbf{H}$, is an $n \times n$ matrix. If we deleted the i^{th} observation when estimating the model, but still asked for a prediction at $\mathbf{x}_i$, we'd get a different, $n \times (n-1)$ matrix, say $\mathbf{H}^{(-i)}$. This in turn would lead to a new fitted value:

$$\hat{m}^{(-i)}(\mathbf{x}_i) = \frac{(\mathbf{Hy})_i - \mathbf{H}_{ii}y_i}{1-\mathbf{H}_{ii}} \tag{5.30}$$

Basically, this is saying we can take the old fitted value, and then subtract off the part of it which came from having included the observation y_j in the first place. Because each row of the hat matrix has to add up to 1, we need to include the denominator.

The **leave-one-out residual** is the difference between this and y_i:

$$e_i^{(-i)} \equiv y_i - \hat{m}^{(-i)}(\mathbf{x}_i) \tag{5.31}$$

That is, this is how far off the model's prediction of y_i would be if it didn't actually get to see y_i during the estimation, but had to honestly predict it.

Leaving out the data point i would give us an MSE of $\hat{\sigma}^2_{(-i)}$, and a little work says that

$$t_i \equiv \frac{e_i^{(-i)}}{\hat{\sigma}_{(-i)}\sqrt{1+\mathbf{x}_i^T(\mathbf{x}_{(-i)}^T\mathbf{x}_{(-i)})^{-1}\mathbf{x}_i}} \; t_{n-p-1} \tag{5.32}$$

These are called the **cross-validated**, or **jackknife**, or **externally studentized**, residuals. (Some people use the name "studentized residuals" only for

these, calling the others the "standardized residuals".) Fortunately, we can compute this without having to actually re-run the regression:

$$t_i = \frac{e_i^{(-i)}}{\hat{\sigma}_{(-i)}\sqrt{1+\mathbf{x}_i^T(\mathbf{x}_{(-i)}^T\mathbf{x}_{(-i)})^{-1}\mathbf{x}_i}} \tag{5.33}$$

$$= \frac{e_i}{\hat{\sigma}_{(-i)}\sqrt{1-H_{ii}}} \tag{5.34}$$

$$= r_i\sqrt{\frac{n-p-1}{n-p-r_i^2}} \tag{5.35}$$

5.4.2 Cook's Distance

Omitting point i will generally change all of the fitted values, not just the fitted value at that point. We go from the vector of predictions $\widehat{\mathbf{m}}$ to $\widehat{\mathbf{m}}^{(-i)}$. How big a change is this? It's natural (by this point!) to use the squared length of the difference vector,

$$\|\widehat{\mathbf{m}} - \widehat{\mathbf{m}}^{(-i)}\|^2 = (\widehat{\mathbf{m}} - \widehat{\mathbf{m}}^{(-i)})^T(\widehat{\mathbf{m}} - \widehat{\mathbf{m}}^{(-i)}) \tag{5.36}$$

To make this more comparable across data sets, it's conventional to divide this by $(p+1)\hat{\sigma}^2$, since there are really only $p+1$ independent coordinates here, each of which might contribute something on the order of $\hat{\sigma}^2$. This is called the **Cook's distance** or **Cook's statistic** for point i:

$$D_i = \frac{(\widehat{\mathbf{m}} - \widehat{\mathbf{m}}^{(-i)})^T(\widehat{\mathbf{m}} - \widehat{\mathbf{m}}^{(-i)})}{(p+1)\hat{\sigma}^2} \tag{5.37}$$

As usual, there is a simplified formula, which evades having to re-fit the regression:

$$D_i = \frac{1}{p+1}e_i^2\frac{H_{ii}}{(1-H_{ii})^2} \tag{5.38}$$

Notice that $H_{ii}/(1-H_{ii})^2$ is a growing function of H_{ii} (Figure 5.4). So this says that the total influence of a point over all the fitted values grows with both its leverage (H_{ii}) and the size of its residual when it is included (e_i^2).

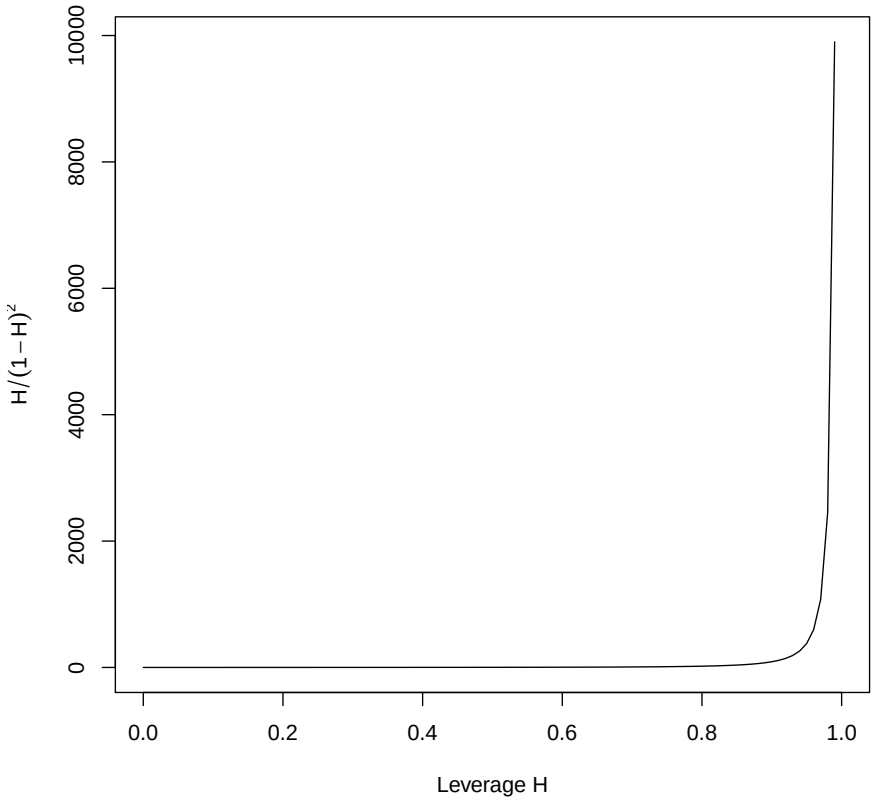
5.4.3 Coefficients

The leave-one-out idea can also be applied to the coefficients. Writing $\widehat{\beta}^{(-i)}$ for the vector of coefficients we get when we drop the i^{th} data point. One can show (**?**, p. 268) that

$$\widehat{\beta}^{(-i)} = \widehat{\beta} - \frac{(\mathbf{x}^T\mathbf{x})^{-1}\mathbf{x}_i^T e_i}{1-H_{ii}} \tag{5.39}$$

Cook's distance can actually be computed from this, since the change in the vector of fitted values is $\mathbf{x}(\widehat{\beta}^{(-i)} - \widehat{\beta})$, so

$$D_i = \frac{((\widehat{\beta}^{(-i)} - \widehat{\beta})^T\mathbf{x}^T\mathbf{x}(\widehat{\beta}^{(-i)} - \widehat{\beta})}{(p+1)\hat{\sigma}^2} \tag{5.40}$$




```
curve(x/(1 - x)^2, from = 0, to = 1, xlab = "Leverage H", ylab = expression(H/(1 -
    H)^2))
```

FIGURE 5.4: *Illustration of the function* $H/(1-H)^2$ *relating leverage* H *to Cook's distance. Notice that leverage must be* ≥ 0 *and* ≤ 1*, so this is the whole relevant range of the curve.*

5.4.4 Leave-More-Than-One-Out

Sometimes, whole clusters of nearby points might be potential outliers. In such cases, removing just one of them might change the model very little, while removing them all might change it a great deal. Unfortunately there are $\binom{n}{k} = O(n^k)$ groups of k points you could consider deleting at once, so while looking at all leave-one-out results is feasible, looking at all leave-two- or leave-ten- out results is not. Instead, you have to think.

5.5 Practically, and with R

We have three ways of looking at whether points are outliers:

1. We can look at their leverage, which depends only on the value of the predictors.

2. We can look at their studentized residuals, either ordinary or cross-validated, which depend on how far they are from the regression line.

3. We can look at their Cook's statistics, which say how much removing each point shifts all the fitted values; it depends on the product of leverage and residuals.

The model assumptions don't put any limit on how big the leverage can get (just that it's ≤ 1 at each point) or on how its distributed across the points (just that it's got to add up to $p+1$). Having most of the leverage in a few super-inferential points doesn't break the model, exactly, but it should make us worry.

The model assumptions *do* say how the studentized residuals should be distributed. In particular, the cross-validated studentized residuals should follow a t distribution. This is something we can test, either for specific points which we're worried about (say because they showed up on our diagnostic plots), or across all the points[2].

Because Cook's distance is related to how much the parameters change, the theory of confidence ellipsoids can be used to get some idea of how big a D_i is worrying[3]. Cook's original rule-of-thumb translates into worrying when $(p+1)D_i$ is bigger than about $\chi^2_{p+1}(0.1)$, though the 0.1 is arbitrary[4]. However, this is not really a hypothesis test.

[2]Be careful about testing all the points. If you use a size α test and everything is fine, you'd see about αn rejections. A good, if not necessarily optimal, way to deal with this is to lower the threshold to α/n for each test — another example of the Bonferroni correction

[3]Remember we saw that for large n, $(\widehat{\beta}-\beta)^T \mathbf{\Sigma}^{-1} (\widehat{\beta}-\beta) \sim \chi^2_{p+1}$, where $\mathbf{\Sigma}$ is the variance matrix of the coefficient estimates. But that's $\sigma^2(\mathbf{x}^T\mathbf{x})^{-1}$, so we get $\sigma^{-2}(\widehat{\beta}-\beta)^T\mathbf{x}^T\mathbf{x}(\widehat{\beta}-\beta) \sim \chi^2_{p+1}$. Now compare with Eq. 5.40.

[4]More exactly, he used an F distribution to take account of small-n uncertainties in $\hat{\sigma}^2$, and suggested worrying when D_i was bigger than $F_{p+1,n-p-1}(0.1)$. This will come to the same thing for large n.

5.5.1 In R

Almost everything we've talked — leverages, studentized residuals, Cook's statistics — can be calculated using the `influence` function. However, there are more user-friendly functions which call that in turn, and are probably better to use.

Leverages come from the 'hatvalues' function, or from the 'hat' component of what 'influence' returns:

```
mob.lm <- lm(Mobility ~ Commute, data = mobility)
hatvalues(mob.lm)
influence(mob.lm)$hat
```

The standardized, or internally-studentized, residuals r_i are available with `rstandard`:

```
rstandard(mob.lm)
residuals(mob.lm)/sqrt(1 - hatvalues(mob.lm))
```

The cross-validated or externally-studentized residuals t_i are available with `rstudent`:

```
rstudent(mob.lm)
```

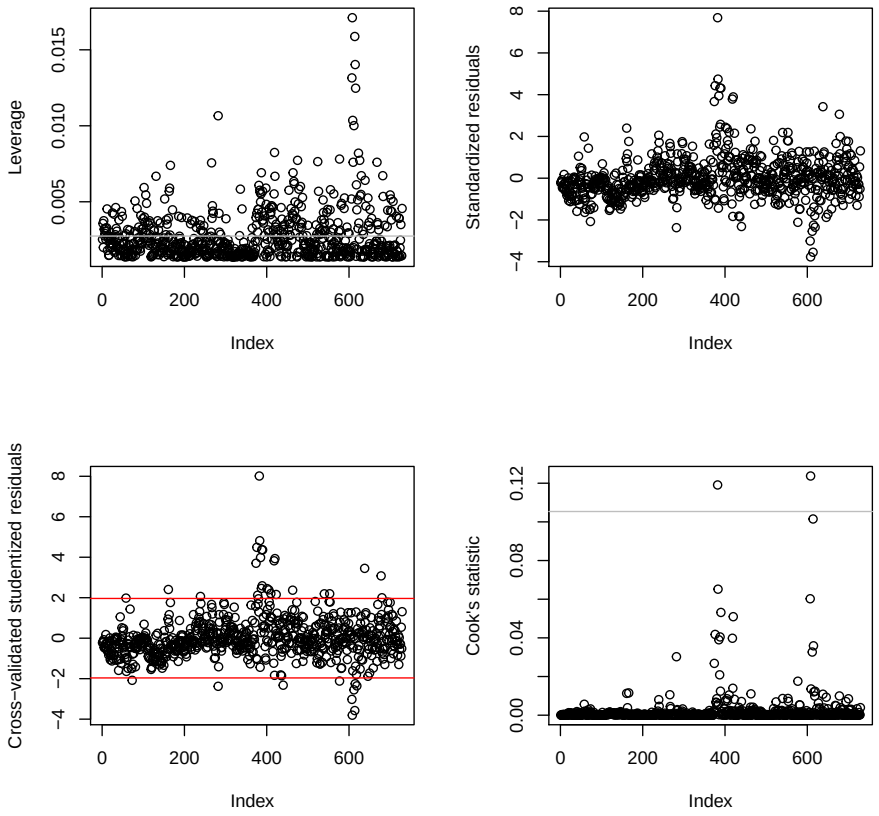
Cook's statistic is calculated with `cooks.distance`:

```
cooks.distance(mob.lm)
```

Often the most useful thing to do with these is to plot them, and look at the most extreme points. (One might also rank them, and plot them against ranks.) Figure 5.5 does so. The standardized and studentized residuals can also be put into our usual diagnostic plots, since they should average to zero and have constant variance when plotted against the fitted values or the predictors. (I omit that here because in this case, $1/\sqrt{1-H_{ii}}$ is sufficiently close to 1 that it makes no visual difference.)

We can now look at exactly which points have the extreme values, say the 10 most extreme residuals, or largest Cook's statistics:

```
mobility[rank(-abs(rstudent(mob.lm)), ) <= 10, ]
##       X       Name   Mobility State Commute  Longitude Latitude
## 374 375     Linton 0.29891303    ND   0.646 -100.16075 46.31258
## 376 378 Carrington 0.33333334    ND   0.656  -98.86684 47.59698
## 382 384     Bowman 0.46969697    ND   0.648 -103.42526 46.33993
## 383 385     Lemmon 0.35714287    ND   0.704 -102.42011 45.96558
## 385 388 Plentywood 0.31818181    MT   0.681 -104.65381 48.64743
## 388 391  Dickinson 0.32920793    ND   0.659 -102.61354 47.32696
## 390 393  Williston 0.33830845    ND   0.702 -103.33987 48.25441
## 418 422     Miller 0.31506848    SD   0.697  -99.27758 44.53313
## 420 424 Gettysburg 0.32653061    SD   0.729 -100.19547 45.05100
## 608 618       Nome 0.04678363    AK   0.928 -162.03012 64.47514
```



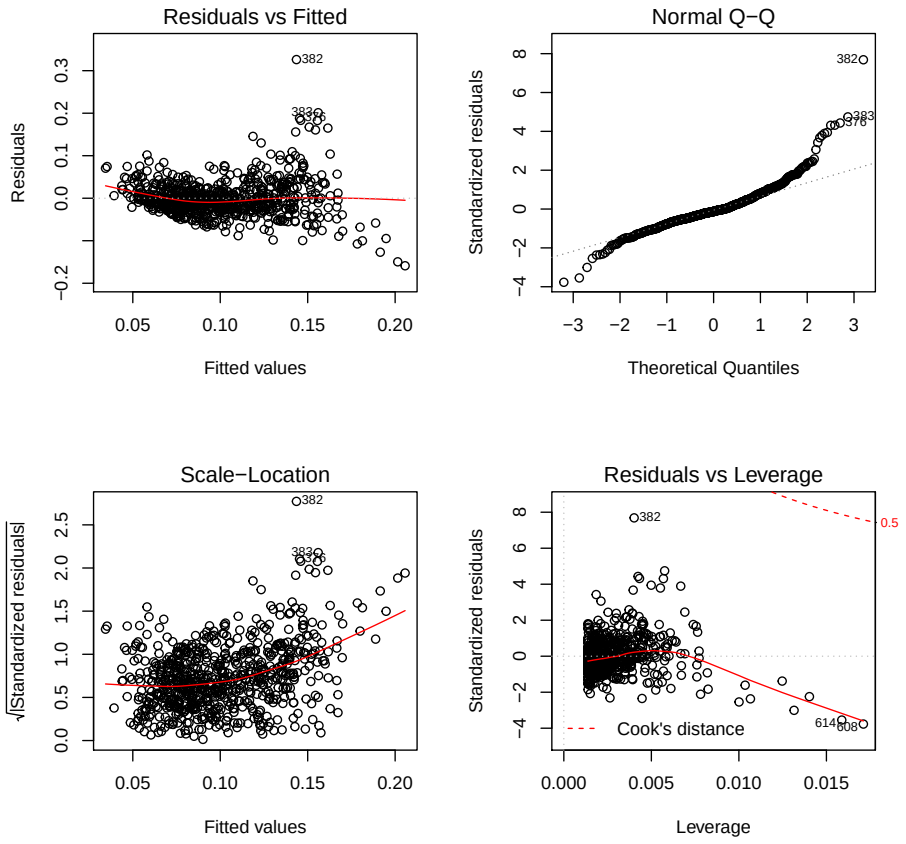

```
par(mfrow = c(2, 2))
mob.lm <- lm(Mobility ~ Commute, data = mobility)
plot(hatvalues(mob.lm), ylab = "Leverage")
abline(h = 2/nrow(mobility), col = "grey")
plot(rstandard(mob.lm), ylab = "Standardized residuals")
plot(rstudent(mob.lm), ylab = "Cross-validated studentized residuals")
abline(h = qt(0.025, df = nrow(mobility) - 2), col = "red")
abline(h = qt(1 - 0.025, df = nrow(mobility) - 2), col = "red")
plot(cooks.distance(mob.lm), ylab = "Cook's statistic")
abline(h = qchisq(0.1, 2)/2, col = "grey")
```

FIGURE 5.5: *Leverages, two sorts of standardized residuals, and Cook's distance statistic for each point in a basic linear model of economic mobility as a function of the fraction of workers with short commutes. The horizontal line in the plot of leverages shows the average leverage. The lines in studentized residual plot shows a 95% t-distribution sampling interval. (What is the grey line in the plot of Cook's distances?) Note the clustering of extreme residuals* and *leverage around row 600, and another cluster of points with extreme residuals around row 400.*

```
mobility[rank(-abs(cooks.distance(mob.lm))) <= 10, ]
##       X       Name   Mobility State Commute  Longitude Latitude
## 376 378 Carrington 0.33333334    ND   0.656  -98.86684 47.59698
## 382 384     Bowman 0.46969697    ND   0.648 -103.42526 46.33993
## 383 385     Lemmon 0.35714287    ND   0.704 -102.42011 45.96558
## 388 391  Dickinson 0.32920793    ND   0.659 -102.61354 47.32696
## 390 393  Williston 0.33830845    ND   0.702 -103.33987 48.25441
## 418 422     Miller 0.31506848    SD   0.697  -99.27758 44.53313
## 420 424 Gettysburg 0.32653061    SD   0.729 -100.19547 45.05100
## 607 617   Kotzebue 0.06451613    AK   0.864 -159.43781 67.02818
## 608 618       Nome 0.04678363    AK   0.928 -162.03012 64.47514
## 614 624     Bethel 0.05186386    AK   0.909 -158.38213 61.37712
```

5.5.2 `plot`

We have not used the `plot` function on an `lm` object yet. This is because most of what it gives us is in fact related to residuals (Figure 5.6). The first plot is of residuals versus fitted values, plus a smoothing line, with extreme residuals marked by row number. The second is a Q-Q plot of the standardized residuals, again with extremes marked by row number. The third shows the square root of the absolute standardized residuals against fitted values (ideally, flat); the fourth plots standardized residuals against leverage, with contour lines showing equal values of Cook's distance. There are many options, described in `help(plot.lm)`.




```
par(mfrow = c(2, 2))
plot(mob.lm)
par(mfrow = c(1, 1))
```

FIGURE 5.6: *The basic* `plot` *function applied to our running example model.*

5.6 Responses to Outliers

There are essentially three things to do when we're convinced there are outliers: delete them; change the model; or change how we estimate.

5.6.1 Deletion

Deleting data points should never be done lightly, but it is sometimes the right thing to do.

The best case for removing a data point is when you have good reasons to think it's just wrong (and you have no way to fix it). Medical records which give a patient's blood pressure as 0, or their temperature as 200 degrees, are just impossible and have to be errors. Those points aren't giving you useful information about the process you're studying, so getting rid of them makes sense.

The next best case is if you have good reasons to think that the data point isn't *wrong*, exactly, but belongs to a different phenomenon or population from the one you're studying. (You're trying to see if a new drug helps cancer patients, but you discover the hospital has included some burn patients and influenza cases as well.) Or the data point does belong to the right population, but also somehow to another one which isn't what you're interested in right now. (All of the data is on cancer patients, but some of them were also sick with the flu.) You should be careful about that last, though. (After all, some proportion of future cancer patients are also going to have the flu.)

The next best scenario after that is that there's nothing quite so definitely wrong about the data point, but it just looks really weird compared to all the others. Here you are really making a judgment call that either the data really are mistaken, or not from the right population, but you can't put your finger on a concrete reason why. The rules-of-thumb used to identify outliers, like "Cook's distance shouldn't be too big", or "Tukey's rule"[5], are at best of this sort. It is always more satisfying, and more reliable, if investigating how the data were gathered lets you turn cases of this sort into one of the two previous kinds.

The least good case for getting rid of data points which isn't just bogus is that you've got a model which almost works, and would work a lot better if you just get rid of a few stubborn points. This is really a sub-case of the previous one, with added special pleading on behalf of your favorite model. You are here basically trusting your model more than your data, so it had better be either a really good model or really bad data.

Beyond this, we get into what can only be called ignoring inconvenient facts so that you get the answer you want.

[5] Which flags any point more than 1.5 times the inter-quartile range above the third quartile, or below the first quartile, on any dimension.

5.6.2 Changing the Model

Outliers are points that break a pattern. This can be because the points are bad, or because we made a bad guess about the pattern. Figure 5.7 shows data where the cloud of points on the right are definite outliers for any linear model. But I drew those points following a quadratic model, and they fall perfectly along it (as they should). Deleting them, in order to make a linear model work better, would have been short-sighted at best.

The moral of Figure 5.7 is that data points can look like outliers because we're looking for the wrong pattern. If when we find apparent outliers and we can't convince ourselves that data is erroneous or irrelevant, we should consider changing our model, before, or as well as, deleting them.

5.6.3 Robust Linear Regression

A final alternative is to change how we estimate our model. Everything we've done has been based on ordinary least-squares (OLS) estimation. Because the squared error grows very rapidly with the error, OLS can be very strongly influenced by a few large "vertical" errors[6]. We might, therefore, consider using not a different statistical model, but a different method of estimating its parameters. Estimation techniques which are less influenced by outliers in the residuals than OLS are called **robust estimators**, or (for regression models) **robust regression**.

Usually (though not always), robust estimation, like OLS, tries to minimize[7] some average of a function of the errors:

$$\tilde{\beta} = \text{argmin}_{\mathbf{b}} \frac{1}{n} \sum_{i=1}^{n} \rho(y_i - \mathbf{x}_i \mathbf{b}) \tag{5.41}$$

Different choices of ρ, the **loss function**, yield different estimators. $\rho(u) = |u|$ is **least absolute deviation** (LAD) estimation. $\rho(u) = u^2$ is OLS again. A popular compromise is to use **Huber's** loss function.

$$\rho(u) = \begin{cases} u^2 & |u| \leq c \\ 2c|u| - c^2 & |u| \geq c \end{cases} \tag{5.42}$$

Notice that Huber's loss looks like squared error for small errors, but like absolute error for large errors. Huber's loss is designed to be continuous at c, and have a continuous first derivative there as well (which helps with optimization). We need to pick the scale c at which it switches over from acting like squared error to acting like absolute error; this is usually done using a robust estimate of the noise standard deviation σ.

[6]Suppose there are 100 data points, and we start with parameter values where $e_1 > 10$, while e_2 through $e_1 00 = 0$. Changing to a new parameter value where $e_i = 1$ for all i actually reduces the MSE, even though it moves us away from perfectly fitting 99% of the data points.

[7]Hence the name "M-estimators".

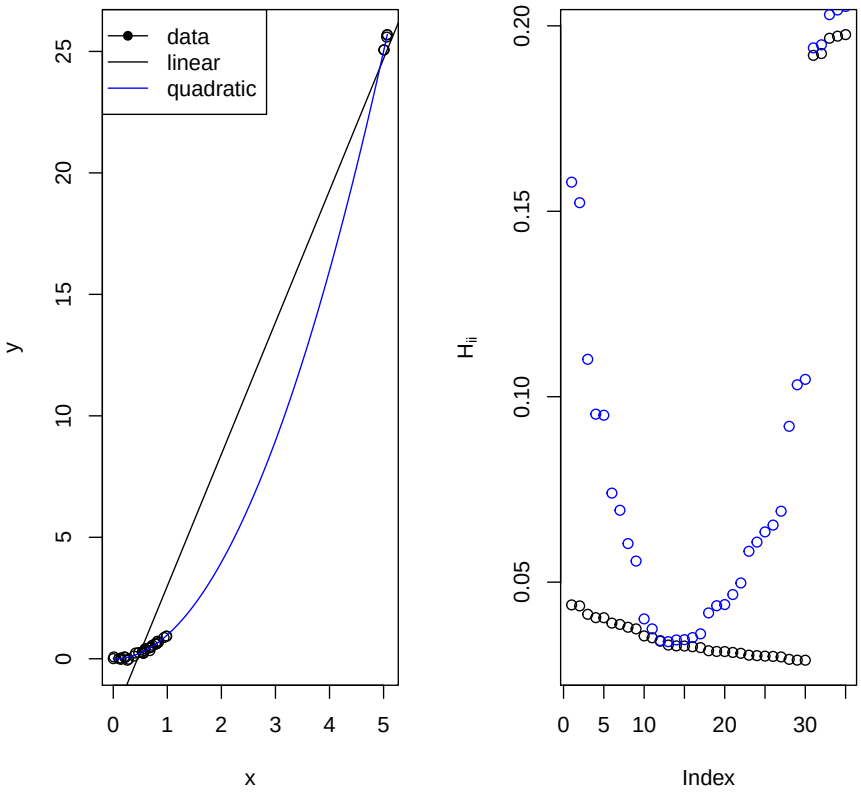


FIGURE 5.7: *The points in the upper-right are outliers for any linear model fit through the main body of points, but dominate the line because of their very high leverage; they'd be identified as outliers. But all points were generated from a quadratic model.*

Robust estimation with Huber's loss can be conveniently done with the `rlm` function in the `MASS` package, which, as the name suggests, is designed to work very much like `lm`.

```
library(MASS)
summary(rlm(Mobility ~ Commute, data = mobility))
##
## Call: rlm(formula = Mobility ~ Commute, data = mobility)
## Residuals:
##       Min        1Q    Median        3Q       Max
## -0.148719 -0.019461 -0.002341  0.021093  0.332347
##
## Coefficients:
##             Value   Std. Error t value
## (Intercept)  0.0028  0.0043     0.6398
## Commute      0.2077  0.0091    22.7939
##
## Residual standard error: 0.0293 on 727 degrees of freedom
```

Robust linear regression is designed for the situation where it's still true that $Y = X\beta + \epsilon$, but the noise ϵ is not very close to Gaussian, and indeed is sometimes "contaminated" by wildly larger values. It does nothing to deal with non-linearity, or correlated noise, or even some points having excessive leverage because we're insisting on a linear model.

Chapter 6

Model Selection

6.1 Generalization and Optimism

We estimated our model by minimizing the mean squared error on our data:

$$\widehat{\beta} = \mathrm{argmin}_{\mathbf{b}} \frac{1}{n}(\mathbf{y} - \mathbf{x}\mathbf{b})^T(\mathbf{y} - \mathbf{x}\mathbf{b})$$

Different linear models will amount to different choices of the design matrix $\mathbf{x}$ — we add or drop variables, we add or drop interactions or polynomial terms, etc., and this adds or removes columns from the design matrix. We might consider doing selecting among models themselves by minimizing the MSE. This is a very bad idea, for a fundamental reason:

> Every model is too optimistic about how well it will actually predict.

Let's be very clear about what it would mean to predict well. The most challenging case would be that we see a new *random* point, with predictor values $X_1, \ldots X_p$ and response Y, and our *old* $\widehat{\beta}$ has a small expected squared error:

$$\mathbb{E}\left[\left(Y - \left(\hat{\beta}_0 + \sum_{j=1}^{p} X_j \hat{\beta}_j\right)\right)^2\right]$$

Here both Y and the X's are random (hence the capital letters), so we might be asking the model for a prediction at a point it never saw before. (Of course if we have multiple identically distributed (X, Y) pairs, say q of them, the expected MSE over those q points is just the same as the expected squared error at one point.)

An easier task would be to ask the model for predictions at the *same* values of the predictor variables as before, but with different random noises. That is, we fit the model to

$$\mathbf{Y} = \mathbf{x}\beta + \epsilon$$

and now Tyche reach into her urn and gives us

$$\mathbf{Y}' = \mathbf{x}\beta + \epsilon'$$

where ϵ and ϵ' are independent but identically distributed. The design matrix is the same, the true parameters β are the same, but the noise is different. We now want to see if the coefficients we estimated from $(\mathbf{x}, \mathbf{Y})$ can predict $(\mathbf{x}, \mathbf{Y}')$. Since the only thing that's changed is the noise, if the coefficients can't predict well any more, that means that they were really just memorizing the noise, and not actually doing anything useful.

Our out-of-sample expected MSE, then, is

$$\mathbb{E}\left[n^{-1}(\mathbf{Y}' - \mathbf{x}\widehat{\beta})^T(\mathbf{Y}' - \mathbf{x}\widehat{\beta})\right]$$

It will be convenient to break this down into an average over data points, and to abbreviate $\mathbf{x}\widehat{\beta} = \widehat{\mathbf{m}}$, the vector of fitted values. Notice that since the predictor variables and the coefficients aren't changing, our predictions are the same both in and out of sample — at point i, we will predict $\widehat{m}_i$.

In this notation, then, the expected out-of-sample MSE is

$$\mathbb{E}\left[\frac{1}{n}\sum_{i=1}^{n}(Y_i' - \widehat{m}_i)^2\right]$$

We'll compare this to the expected in-sample MSE,

$$\mathbb{E}\left[\frac{1}{n}\sum_{i=1}^{n}(Y_i - \widehat{m}_i)^2\right]$$

Notice that $\widehat{m}_i$ is a function of Y_i (among other things), so those are dependent random variables, while $\widehat{m}_i$ and Y_i' are completely statistically independent.

Break this down term by term. What's the expected value of the i^{th} in-sample squared error?

$$\begin{aligned}
\mathbb{E}\left[(Y_i - \hat{m}_i)^2\right] &= \mathrm{Var}\left[Y_i - \hat{m}_i\right] + \left(\mathbb{E}\left[Y_i - \hat{m}_i\right]\right)^2 && (6.1)\\
&= \mathrm{Var}\left[Y_i\right] + \mathrm{Var}\left[\hat{m}_i\right] - 2\mathrm{Cov}\left[Y_i, \hat{m}_i\right] + \left(\mathbb{E}\left[Y_i\right] - \mathbb{E}\left[\hat{m}_i\right]\right)^2 && (6.2)
\end{aligned}$$

The covariance term is not (usually) zero, because, as I just said, $\hat{m}_i$ is a function of, in part, Y_i.

On the other hand, what's the expected value of the i^{th} squared error on new data?

$$\begin{aligned}
\mathbb{E}\left[(Y_i' - \hat{m}_i)^2\right] &= \mathrm{Var}\left[Y_i\prime - \hat{m}_i\right] + \left(\mathbb{E}\left[Y_i' - \hat{m}_i\right]\right)^2 && (6.3)\\
&= \mathrm{Var}\left[Y_i'\right] + \mathrm{Var}\left[\hat{m}_i\right] - 2\mathrm{Cov}\left[Y_i', \hat{m}_i\right] + \left(\mathbb{E}\left[Y_i'\right] - \mathbb{E}\left[\hat{m}_i\right]\right)^2 && (6.4)
\end{aligned}$$

Y_i' is independent of Y_i, but has the same distribution. This tells us that $\mathbb{E}\left[Y_i'\right] = \mathbb{E}\left[Y_i\right]$, $\mathrm{Var}\left[Y_i'\right] = \mathrm{Var}\left[Y_i\right]$, but $\mathrm{Cov}\left[Y_i', \hat{m}_i\right] = 0$. So

$$\begin{aligned}
\mathbb{E}\left[(Y_i' - \hat{m}_i)^2\right] &= \mathrm{Var}\left[Y_i\right] + \mathrm{Var}\left[\hat{m}_i\right] + \left(\mathbb{E}\left[Y_i\right] - \mathbb{E}\left[\hat{m}_i\right]\right)^2 && (6.5)\\
&= \mathbb{E}\left[(Y_i - \hat{m}_i)^2\right] + 2\mathrm{Cov}\left[Y_i, \hat{m}_i\right] && (6.6)
\end{aligned}$$

Averaging over data points,

$$\mathbb{E}\left[\frac{1}{n}\sum_{i=1}^{n}(Y_i' - \widehat{m}_i)^2\right] = \mathbb{E}\left[\frac{1}{n}\sum_{i=1}^{n}(Y_i - \widehat{m}_i)^2\right] + \frac{2}{n}\sum_{i=1}^{n}\mathrm{Cov}\left[Y_i, \hat{m}_i\right]$$

Clearly, we need to get a handle on that sum of covariances.
For a linear model, though, $\mathrm{Cov}\left[Y_i, \hat{m}_i\right] = \sigma^2 H_{ii}$. So, for linear models,

$$\mathbb{E}\left[\frac{1}{n}\sum_{i=1}^{n}(Y_i' - \widehat{m}_i)^2\right] = \mathbb{E}\left[\frac{1}{n}\sum_{i=1}^{n}(Y_i - \widehat{m}_i)^2\right] + \frac{2}{n}\sigma^2 \mathrm{tr}\left(\mathbf{H}\right)$$

and we know that with p predictors and one intercept, $\mathrm{tr}\left(\mathbf{H}\right) = p+1$ (Homework 5). Thus, for linear models,

$$\mathbb{E}\left[\frac{1}{n}\sum_{i=1}^{n}(Y_i' - \widehat{m}_i)^2\right] = \mathbb{E}\left[\frac{1}{n}\sum_{i=1}^{n}(Y_i - \widehat{m}_i)^2\right] + \frac{2}{n}\sigma^2(p+1)$$

Of course, we don't actually know the *expectation* on the right-hand side, but we do have a sample estimate of it, which is the in-sample MSE. If the law of large numbers is still our friend,

$$\mathbb{E}\left[\frac{1}{n}\sum_{i=1}^{n}(Y_i' - \widehat{m}_i)^2\right] \approx \frac{1}{n}\sum_{i=1}^{n}(Y_i - \widehat{m}_i)^2 + \frac{2}{n}\sigma^2(p+1)$$

The second term on the right, $(2/n)\sigma^2(p+1)$, is the **optimism** of the model — the amount by which its in-sample MSE systematically under-estimates its true expected squared error. Notice that this:

- Grows with σ^2: more noise gives the model more opportunities to seem to fit well by capitalizing on chance.
- Shrinks with n: at any fixed level of noise, more data makes it harder to pretend the fit is better than it really is.
- Grows with p: every extra parameter is another control which can be adjusted to fit to the noise.

Minimizing the in-sample MSE completely ignores the bias from optimism, so it is guaranteed to pick models which are too large and predict poorly out of sample. If we could calculate the optimism term, we could at least use an unbiased estimate of the true MSE on new data.
Of course, we do not actually know σ^2.

6.2 Mallow's C_p Statistic

The Mallows C_p statistic just substitutes in a feasible estimator of σ^2, which is $\hat{\sigma}^2$ *from the largest model we consider*. This will be an unbiased estimator of σ^2

if the real model is smaller (contains a strict subset of the predictor variables), but not vice versa[1].

That is, for a linear model with $p+1$ coefficients fit by OLS,

$$C_p \equiv= \frac{1}{n}\sum_{i=1}^{n}(Y_i - \widehat{m}_i)^2 + \frac{2}{\hat{\sigma}^2}n(p+1) \tag{6.7}$$

The selection rule is to pick the model which minimizes C_p.

We can think of C_p as having two parts,

$$C_p = MSE + (\text{penalty})$$

From one point of view, the penalty is just an estimate of the bias. From another point of view, it's a cost we're imposing on models for having extra parameters. Every new parameter has got to pay that cost by reducing the MSE by at least a certain amount; if it doesn't, the extra parameter isn't worth it.

(Before this, we've only been dealing with *one* model, so we've not had to distinguish carefully between the in-sample MSE and the maximum likelihood estimate of σ^2. With multiple models floating around, though, each can have its own MSE, but there is only one true σ^2, and we need *an* estimate of it.)

For comparing models, we really care about differences:

$$\Delta C_p = MSE_1 - MSE_2 + \frac{2}{n}\hat{\sigma}^2(p_1 - p_2) \tag{6.8}$$

(The extra term for the intercept, being common to both models, doesn't contribute.)

Alternate form of C_p You will find many references which define C_p somewhat differently:

$$\frac{nMSE}{\hat{\sigma}^2} - n + 2p \tag{6.9}$$

and say that the optimal value is close to p, not close to 0. To see that this selects exactly the same models as the rule given above, take a difference between two models, with MSE's MSE_1, MSE_2 and p_1, p_2 predictors. We get

$$\frac{n(MSE_1 - MSE_2)}{\hat{\sigma}^2} + 2(p_1 - p_2)$$

Dividing by n and multiplying by $\hat{\sigma}^2$ gives us back Eq. 6.8. There are reasons to assert that Eq. 6.9 should indeed be close to p for the right model (if the Gaussian noise assumption holds), but Eq. 6.7 is a good estimate of the out-of-sample error, and a good model selection rule, much more broadly.

[1]This assumes the largest model must contain the truth!

6.3 R^2 and Adjusted R^2

Recall that

$$R^2 = 1 - \frac{MSE}{s_Y^2}$$

Picking a model by maximizing R^2 is thus equivalent to picking a model by minimizing MSE. It is therefore stupid for exactly the same reasons that minimizing MSE across models is stupid.

Recall that the adjusted R^2 is

$$R^2_{adj} = 1 - \frac{MSE \frac{n}{n-p-1}}{s_Y^2}$$

That is, it's R^2 with the unbiased estimator of σ^2. Maximizing adjusted R^2 therefore corresponds to minimizing that unbiased estimator. What does that translate to?

$$\begin{aligned} MSE\frac{n}{n-p-1} &= MSE\frac{1}{1-(p+1)/n} && (6.10)\\ &\approx MSE\left(1+\frac{p+1}{n}\right) && (6.11)\\ &= MSE + MSE\frac{p+1}{n} && (6.12) \end{aligned}$$

where the approximation becomes exact as $n \to \infty$ with p fixed[2]. Even for the completely right model, where MSE is a consistent estimator of $\hat{\sigma}^2$, the correction or penalty is only half as big as we've seen it should be. Selecting models using adjusted R^2 is not completely stupid, as maximizing R^2 is, but it is still not going to work very well.

6.4 Akaike Information Criterion (AIC)

The great Japanese statistician Hirotugu Akaike proposed a famous model selection rule which also has the form of "in-sample performance plus penalty". What has come to be called the **Akaike information criterion** (AIC) is

$$AIC(S) \equiv L_S - \dim(S)$$

where L_S is the log likelihood of the model S, evaluated at the maximum likelihood estimate, and $\dim(S)$ is the dimension of S, the number of adjustable parameters it has. Akaike's rule is to pick the model which maximizes AIC[3].

[2] Use the binomial theorem to expand $1/(1-u)$ as $1+u+u^2+\ldots$, and truncate the series at first order. (If u is small, u^2 is tiny, and the higher powers microscopic.)

[3] Actually, in his original paper (**?**), he proposed using *twice* this, to simplify some calculations involving chi-squared distributions. Many subsequent authors have since kept the factor of 2, which of course will not change which model is selected. Also, some authors define AIC as negative of this, and then minimize it; again, clearly the same thing.

The reason for this definition is that Akaike showed AIC/n is an unbiased estimate of the expected log-probability the estimated parameters will give to a new data point which it hasn't seen before, if the model is right. This is the natural counterpart of expected squared error for more general distributions than the Gaussian. IF we do specialize to linear-Gaussian models, then we've seen that

$$L = -\frac{n}{2}(1 + \log 2\pi) - \frac{n}{2}\log MSE$$

and the dimension of the model is $p + 2$ (because σ^2 is also an adjustable parameter). Notice that $-\frac{n}{2}(1+\log 2\pi)$ doesn't involve the parameters at all. If we compare AICs for two models, with mean squared errors in-sample of MSE_1 and MSE_2, and one with p_1 predictors and the other with p_2, the difference in AICs will be

$$\Delta AIC = -\frac{n}{2}\log MSE_1 + \frac{n}{2}\log MSE_2 - (p_1 - p_2)$$

To relate this to C_p, let's write $MSE_2 = MSE_1 + \Delta MSE$. Then

$$\begin{aligned} \Delta AIC &= -\frac{n}{2}\log MSE_1 + \frac{n}{2}\log MSE_1\left(1 + \frac{\Delta MSE}{MSE_1}\right) - (p_1 - p_2) && (6.13)\\ &= -\frac{n}{2}\log\left(1 + \frac{\Delta MSE}{MSE_1}\right) - (p_1 - p_2) && (6.14) \end{aligned}$$

Now let's suppose that model 1 is actually the correct model, so $MSE_1 = \hat{\sigma}^2$, and that ΔMSE is small compared to $\hat{\sigma}^2$, so[4]

$$\begin{aligned} \Delta AIC &\approx -\frac{n}{2}\frac{\Delta MSE}{\hat{\sigma}^2} - (p_1 - p_2) && (6.15)\\ \frac{-2\hat{\sigma}^2}{n}\Delta AIC &\approx \Delta MSE + \frac{2}{n}\hat{\sigma^2}(p_1 - p_2) = \Delta C_p && (6.16) \end{aligned}$$

So, if one of the models we're looking at is actually the correct model, and the others aren't too different from it, picking by maximizing AIC will give the same answer as picking by minimizing C_p.

Other Uses of AIC AIC can be applied whenever we have a likelihood. It is therefore used for tasks like comparing models of probability distributions, or predictive models where the whole distribution is important. C_p, by contrast, really only makes sense if we're trying to do regression and want to use squared error.

Why $-\dim(S)$? Akaike had a truly brilliant argument for subtracting a penalty equal to the number of parameters from the log-likelihood, which is too pretty not to at least sketch here.

[4] Taylor expand $\log 1 + u$ around 1 to get $\log 1 + u \approx u$, for u close to 0.

Generically, say that the parameter vector is θ, and its true value is θ^*. (For linear regression with Gaussian noise, θ consists of all $p+1$ coefficients plus σ^2.) The length of this vector, which is $\dim(S)$, is let's say d. (For linear regression with Gaussian noise, $d = p + 2$.) The maximum likelihood estimate is $\hat{\theta}$. We know that the derivative of the likelihood is zero at the MLE:

$$\nabla L(\hat{\theta}) = 0$$

Let's do a Taylor series expansion of $\nabla L(\theta)$ around the true parameter value θ^*:

$$\nabla L(\theta) = \nabla L(\theta^*) + (\theta - \theta^*)\nabla\nabla L(\theta^*)$$

Here $\nabla\nabla L(\theta^*)$ is the $d \times d$ matrix of second partial derivatives of L, evaluated at θ^*. This is called the **Hessian**, and would traditionally be written $\mathbf{H}$, but that would lead to confusion with the hat matrix, so I'll call it $\mathbf{K}$. Therefore the Taylor expansion for the gradient of the log-likelihood is

$$\nabla L(\theta) = \nabla L(\theta^*) + (\theta - \theta^*)\mathbf{K}$$

Applied to the MLE,

$$\mathbf{0} = \nabla L(\theta^*) + (\hat{\theta} - \theta^*)\mathbf{K}$$

or

$$\hat{\theta} = \theta^* - K^{-1}\nabla L(\theta^*)$$

What is the *expected* log-likelihood, on new data, of $\hat{\theta}$? Call this expected log-likelihood ℓ (using a lower-case letter to indicate that it is non-random). Doing another Taylor series,

$$\ell(\theta) \approx \ell(\theta^*) + (\theta - \theta^*)^T\nabla\ell(\theta^*) + \frac{1}{2}(\theta - \theta^*)^T\nabla\nabla\ell(\theta^*)(\theta - \theta^*)$$

However, it's not hard to show that the expected log-likelihood is in general maximized by the true parameters, so $\nabla\ell(\theta^*) = 0$. (The same argument also shows $\mathbb{E}\left[\nabla L(\theta^*)\right] = 0$.) Call the Hessian in this Taylor expansion $\mathbf{k}$. (Again, notice the lower-case letter for a non-random quantity.) We have

$$\ell(\theta) \approx \ell(\theta^*) + \frac{1}{2}(\theta - \theta^*)^T\mathbf{k}(\theta - \theta^*)$$

Apply this to the MLE:

$$\ell(\hat{\theta}) \approx \ell(\theta^*) + \frac{1}{2}\nabla L(\theta^*)\mathbf{K}^{-1}\mathbf{k}\mathbf{K}^{-1}\nabla L(\theta^*)$$

Taking expectations,

$$\mathbb{E}\left[\ell(\hat{\theta})\right] \approx \ell(\theta^*) + \frac{1}{2}\mathrm{tr}\left(\mathbf{K}^{-1}\mathbf{k}\mathbf{K}^{-1}\mathbf{J}\right)$$

where $\mathrm{Var}\left[\nabla L(\theta^*)\right] = \mathbf{J}$. For large n, $\mathbf{K}$ converges on $\mathbf{k}$, so this simplifies to

$$\mathbb{E}\left[\ell(\hat{\theta})\right] \approx \ell(\theta^*) + \frac{1}{2}\mathrm{tr}\left(\mathbf{k}^{-1}\mathbf{J}\right)$$

This still leaves things in terms of $\ell(\theta^*)$, which of course we don't know, but now we do another Taylor expansion, this time of L around $\hat{\theta}$:

$$L(\theta^*) \approx L(\hat{\theta}) + \frac{1}{2}(\theta^* - \hat{\theta})^T \nabla\nabla L(\hat{\theta})(\theta^* - \hat{\theta})$$

so

$$L(\theta^*) \approx L(\hat{\theta}) + \frac{1}{2}(\mathbf{K}^{-1}\nabla L(\theta^*))^T \nabla\nabla L(\hat{\theta})(\mathbf{K}^{-1}\nabla L(\theta^*))$$

For large n, $\nabla\nabla L(\hat{\theta}) \to \nabla\nabla L(\theta^*) \to \mathbf{k}$. So, again taking expectations,

$$\ell(\theta^*) \approx \mathbb{E}\left[L(\hat{\theta}\right] + \frac{1}{2}\mathrm{tr}\left(\mathbf{k}^{-1}\mathbf{J}\right)$$

Putting these together,

$$\mathbb{E}\left[\ell(\hat{\theta})\right] \approx \mathbb{E}\left[L(\hat{\theta}\right] + \mathrm{tr}\left(\mathbf{k}^{-1}\mathbf{J}\right)$$

An unbiased estimate is therefore

$$L(\hat{\theta}) + \mathrm{tr}\left(\mathbf{k}^{-1}\mathbf{J}\right)$$

Finally, a fundamental result (the "Fisher identity") says that for well-behaved models, *if* the model is correct, then

$$\mathrm{Var}\left[\nabla L(\theta^*)\right] = -\nabla\nabla\ell(\theta^*)$$

or $\mathbf{J} = -\mathbf{k}$. Hence, if the model is correct, our unbiased estimate is just

$$L(\hat{\theta}) - \mathrm{tr}\left(\mathbf{I}\right)$$

and of course $\mathrm{tr}\left(\mathbf{I}\right) = d$.

There, as you'll notice, several steps where we're making a bunch of approximations. Some of these approximations (especially those involving the Taylor expansions) can be shown to be OK asymptotically (i.e., as $n \to \infty$) by more careful math. The last steps, however, where we invoke the Fisher identity, are rather more dubious. (After all, all of the models we're working with can hardly contain the true distribution.) A somewhat more robust version of AIC is therefore to use as the criterion

$$L(\hat{\theta}) + \mathrm{tr}\left(\mathbf{KJ}\right)$$

6.5 Leave-one-out Cross-Validation (LOOCV)

When looking at influential points and outliers, we considered omitting one point from the data set, estimating the model, and then trying to predict that one data point. The **leave-one-out** fitted value for data point i is $\hat{m}_i^{(-i)}$, where

the subscript $(-i)$ indicates that point i was left out in calculating this fit. The **leave-one-out cross-validation score** of the model is

$$LOOCV = \frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{m}_i^{(-i)})^2$$

(Many more old-fashioned regression textbooks look at $nLOOCV$, and call it PRESS, "predictive residual sum of squares".)

The story for cross-validation is pretty compelling: we want to know if our model can generalize to new data, so *see* how well it generalizes to new data. Leaving out each point in turn ensures that that the set of points on which we try to make predictions is just as representative of the whole population as the original sample was. Fortunately, this is one of those cases where a compelling story is actually true: LOOCV is an unbiased estimate of the generalization error.

6.5.1 Short-cut Based on Leverage

Re-estimating the model n times would be seriously time-consuming, but there is fortunately a short-cut:

$$LOOCV = \frac{1}{n}\sum_{i=1}^{n}\left(\frac{Y_i - \hat{m}_i}{1 - H_{ii}}\right)^2$$

The numerator inside the square is just the residual of the model fit to the full data. This gets divided by $1 - H_{ii}$, which is also something we can calculate with just one fit to the model. (The denominator says that the residuals for high-leverage points count more, and those for low-leverage points count less. If the model is going out of its way to match Y_i (high leverage H_{ii}) and it still can't fit it, that's worse than the same sized residual at a point the model doesn't really care about (low leverage).)

The gap between LOOCV and the MSE can be thought of as a penalty, just like with C_p or AIC. The penalty doesn't have such a nice mathematical expression, but it's well-defined and easy for us to calculate.

It also converges to the penalty C_p applies as n grows. To help see this, first observe that the H_{ii} must be getting small. (We know that $\sum_i H_{ii} = p + 1$.) Then[5] $(1 - H_{ii})^{-2} \approx 1 - 2H_{ii}$, and

$$LOOCV \approx \frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{m}_i)^2(1 - 2H_{ii}) \approx MSE + 2\sigma^2 \mathrm{tr}\,(\mathbf{H})$$

Cross-validation with log-likelihood The leave-one-out idea can also be applied for any model where we make a probabilistic prediction. Instead of measuring mean squared error, we measure the negative log probability density

[5] Use the binomial theorem again.

the model assigns to the actual left-out point. (Negative, so that a lower score is still better.) With Gaussian noise, this comes to the same thing as the MSE, of course.

6.5.2 Summing Up C_p, AIC, LOOCV

Under a very broad range of circumstances, there are theorems which say, roughly, the following:

> As $n \to \infty$, the expected out-of-sample MSE of the model picked by leave-one-out cross-validation is close to that of the best model considered.

The condition for these results do *not* require that any of the models considered be true, or that the true model have Gaussian noise or even be linear.

As we've seen, for large n leave-one-out and Mallow's C_p become extremely similar, and will pick the same model, and so will AIC, if one of the models is right. So they will also pick models which predict almost as well as the best of the models we're working with. Since C_p and AIC involve less calculation than leave-one-out, they have advantages when n is large. Against this, there don't seem to be any situations where C_p or AIC pick models with good predictive performance but leave-one-out does not. The best way to think about C_p and AIC is that they are fast approximations to the more fundamental quantity, which is leave-one-out.

On the other hand, one can *also* prove the following:

> As $n \to \infty$, if the true model is among those being compared, LOOCV, C_p and AIC will all tend to pick a *strictly larger* model than the truth.

That is, all three criteria tend to prefer models which are bigger than the true model, even when the true model is available to them. They are "not consistent for model selection".

The problem is that while these methods give unbiased estimates of the generalization error, that doesn't say anything about the variance of the estimates. Models with more parameters have higher variance, and the penalty applied by these methods isn't strong enough to overcome the chance of capitalizing on that variance.

6.6 Other Model Selection Criteria

While many, many other model selection criteria have been proposed, two are particularly important.

6.6.1 k-Fold Cross-Validation

In leave-one-out cross-validation, we omitted each data point in turn, and tried to predict it. K-fold cross-validation is somewhat different, and goes as follows.

- Randomly divide the data into k equally-sized parts, or "folds".
- For each fold
  - Temporarily hold back that fold, calling it the "testing set".
  - Call the other $k-1$ folds, taken together, the "training set".
  - Estimate each model on the training set.
  - Calculate the MSE of each model on the testing set.
- Average MSEs over folds.

We then pick the model with the lowest MSE, averaged across testing sets.

The point of this is just like the point of leave-one-out: the models are compared only on data which they didn't get to see during estimation. Indeed, leave-one-out is the special case of k-fold cross-validation where $k = n$. The disadvantage of doing that is that in leave-one-out, all of the training sets are very similar (they share $n-2$ data points), so averaging over folds does very little to reduce variance. For moderate k — people typically use 5 or 10 — k-fold CV tends to produce very good model selection results.

Like leave-one-out CV, k-fold cross-validation can be applied to any loss function, such as the proportion of cases mis-classified, or negative log-likelihood.

6.6.2 BIC

A more AIC-like criterion is the "Bayesian[6] information criterion" introduced by **?**. The name is quite misleading[7], but irrelevant; it's got the exact same idea of penalizing the log-likelihood with the number of parameters, but using a penalty which gets bigger with n:

$$BIC(S) = L_S - \frac{\log n}{2}\dim(S)$$

This is a stronger penalty than AIC applies, and this has consequences:

> As $n \to \infty$, if the true model is among those BIC can select among, BIC will tend to pick the true model.

Of course there are various conditions attached to this, some of them quite technical, but it's generally true for IID samples, for regression modeling, for many sorts of time series model, etc. Unfortunately, the model selected by BIC will tend to predict less well than the one selected by leave-one-out cross-validation or AIC.

[6] Bayesianism is the idea that we ought to have probabilities for parameter values and for models, and not just for random variables (or, said another way, to treat parameters and models as also random variables), and update those probabilities as we see more events using Bayes's rule)

[7] The truly Bayesian position is not to *select* a model at all, but rather to maintain a probability distribution over all models you think possible.

6.6.3 Stepwise Model Selection

One way to automatically select a model is to begin with the largest model you can, and then prune it, which can be done in several ways:

- Eliminate the least-significant coefficient.
- Pick your favorite model selection criterion, consider deleting each coefficient in turn, and pick the sub-model with the best value of the criterion.

Having eliminated a variable, one then re-estimates the model, and repeats the procedure. Stop when either all the remaining coefficients are significant (under the first option), or nothing can be eliminated without worsening the criterion.

(What I've described is **backwards** stepwise model selection. **Forward** stepwise model selection starts with the intercept-only model and adds variables in the same fashion. There are, naturally, forward-backward hybrids.)

Stepwise model selection is a **greedy** procedure: it takes the move which does the most to immediately improve the criterion, without considering the consequences down the line. There are very, very few situations where it is consistent for model selection, or (in its significance-testing version) where it even does a particularly good job of coming up with predictive models, but it's surprisingly popular.

6.7 Inference after Selection

All of the inferential statistics we have reviewed presumed that our choice of model was completely fixed, and not at all dependent on the data. If different data sets would lead us to use different models, and our data are (partly) random, then which model we're using is also random. This leads to some extra uncertainty in, say, our estimate of the slope on X_1, which is *not* accounted for by our formulas for the sampling distributions, hypothesis tests, confidence sets, etc.

A very common response to this problem, among practitioners, is to ignore it, or at least hope it doesn't matter. This can be OK, if the data-generating distribution forces us to pick one model with very high probability, or if all of the models we might pick are very similar to each other. Otherwise, ignoring it leads to nonsense.

Here, for instance, I simulate 200 data points where the Y variable is a standard Gaussian, and there are 100 independent predictor variables, all also standard Gaussians, independent of each other *and of* Y:

```
n <- 200
p <- 100
y <- rnorm(n)
x <- matrix(rnorm(n * p), nrow = n)
df <- data.frame(y = y, x)
mdl <- lm(y ~ ., data = df)
```

Of the 100 predictors, 4 have t-statistics which are significant at the 0.05 level or less. (The expected number would be 5.) If I select the model using just those variables, I get the following:

```
stars <- 1 + which(coefficients(summary(mdl))[-1, 4] < 0.05)
mdl.2 <- lm(y ~ ., data = df[, c(1, stars)])
summary(mdl.2)
##
## Call:
## lm(formula = y ~ ., data = df[, c(1, stars)])
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -2.76734 -0.57195 -0.03144  0.63081  2.45512
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept) -0.007749   0.069826  -0.111   0.9118
## X37         -0.126228   0.067866  -1.860   0.0644 .
## X60         -0.163841   0.067254  -2.436   0.0157 *
## X63         -0.111425   0.073517  -1.516   0.1312
## X100         0.087094   0.074944   1.162   0.2466
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.9635 on 195 degrees of freedom
## Multiple R-squared:  0.06298,Adjusted R-squared:  0.04376
## F-statistic: 3.276 on 4 and 195 DF,  p-value: 0.01257
```

Notice that final over-all F statistic: it's testing whether including those variables fits better than an intercept-only model, and saying it thinks it does, with a definitely significant p-value. This is the case even though, by construction, the response is *completely independent* of *all* predictors. This is not a fluke: if you re-run my simulation many times, your p-values in the full F test will not be uniformly distributed (as they would be on all 100 predictors), but rather will have a distribution strongly shifted over to the left. Similarly, if we looked at the confidence intervals, they would be much too narrow.

These issues do not go away if the true model isn't "everything is independent of everything else", but rather has some structure. Because we picked the model to predict well on this data, if we then run hypothesis tests on that same data, they'll be too likely to tell us everything is significant, and our confidence intervals will be too narrow. Doing statistical inference on the same data we used to select our model is just broken. It may not always be as spectacularly broken as in my demo above, but it's still broken.

There are three ways around this. One is to pretend the issue doesn't exist; as I said, this is popular, but it's got nothing else to recommend it. Another, which is an area of very active research currently in statistics, is to try to come up with clever technical adjustments to the inferential statistics. The third

approach, which is in many ways the simplest, is to use *different data sets* to select a model and to do inference within the selected model[8].

Data Splitting. Data splitting is (for regression) a very simple procedure:

- Randomly divide your data set into two parts.
- Calculate your favorite model selection criterion for all your candidate models using only the first part of the data. Pick one model as the winner.
- Re-estimate the winner, and calculate all your inferential statistics, using only the other half of the data.

(Division into two equal halves is optional, but usual.)

Because the winning model is statistically independent of the second half of the data, the confidence intervals, hypothesis tests, etc., can treat it as though that model were fixed *a priori*. Since we're only using $n/2$ data points to calculate confidence intervals (or whatever), they will be somewhat wider than if we really had fixed the model in advance and used all n data points, but that's the price we pay for having to select a model based on data.

6.8 R Practicalities

R^2 and adjusted R^2 are calculated by the `summary` function for `lm` objects, if — Heaven forbid – you should ever need them. So, more practically, is the in-sample root mean squared error, using the unbiased estimator:

```
mdl <- lm(something ~ other_things, data = df)
summary(mdl)$r.squared
summary(mdl)$adj.r.squared
summary(mdl)$sigma
```

The un-adjusted MSE is also easily calculated:

```
mean(residuals(mdl)^2)
```

The `AIC` function knows how to work with models produced by `lm`; it uses an alternate definition of AIC which is $-2\times$ the one I gave above (so smaller AIC is preferred). Similarly for the `BIC` function.

The `step` function will do stepwise model selection based on AIC, either forward or backward. Manipulating the arguments also allows for doing BIC. (See the help file.) *Warning:* By default this prints out a lot of information about every model it looks at; consider setting `trace=0`.

For leave-one-out cross-validation, the most straightforward approach is to use the following function:

[8]Technically, there is a fourth possible approach, which is to select the model completely at random, and then do inference within it. This may sound like a joke, but there are actually situations, like testing for a difference in means between high-dimensional vectors, where it's perfectly reasonable.

```
cv.lm <- function(mdl) {
    return(mean((residuals(mdl)/(1 - hatvalues(mdl)))^2))
}
```

For k-fold cross-validation, the easiest option at this stage is to use the `cv.glm` function in the package `boot`. Note that this requires you to fit your model with the `glm` function, not with `lm`, and that you will really only be interested in the `delta` component of what `cv.glm` returns. (See the help file, especially the examples at the end.)

Nobody seems to have written a function for calculating C_p. Here is one.

```
Cp.lm <- function(mdl.list) {
    n <- nobs(mdl.list[[1]])
    DoFs <- sapply(mdl.list, function(mdl) {
        sum(hatvalues(mdl))
    })
    MSEs <- sapply(mdl.list, function(mdl) {
        mean(residuals(mdl)^2)
    })
    biggest <- which.max(DoFs)
    sigma2.hat <- MSEs[[biggest]] * n/(n - DoFs[[biggest]])
    Cp <- MSEs + 2 * sigma2.hat * DoFs/n
    return(Cp)
}


Cp.lm(list(mdl1, mdl2, mdl3))
```

A Demo. We'll do polynomial regression with just one X variable; this way we can keep throwing in as many terms as we need to, in order to make the point. X will be uniformly distributed on the interval $[-2, 2]$, and when we use a q^{th} order polynomial, we'll set

$$Y = \sum_{i=1}^{q} (-1)^q X^q + \epsilon$$

with ϵ having our usual Gaussian distribution with mean 0 and standard deviation 0.1.

Here's code to simulate from the model:

```
sim.poly <- function(n, degree) {
    x <- runif(n, min = -2, max = 2)
    poly.x <- poly(x, degree = degree, raw = TRUE)
    alternating.signs <- rep(c(-1, 1), length.out = degree)
    sum.poly <- poly.x %*% alternating.signs
    y <- x + rnorm(n, 0, 0.1)
    return(data.frame(x = x, y = y))
}
```

And here is code to fit many polynomials to it:

```
poly.fit <- function(df, max.degree) {
    lapply(1:max.degree, function(deg) {
        lm(y ~ poly(x, degree = deg), data = df)
    })
}
```

And to apply multiple selection criteria to a list of models:

```
selectors <- function(mdl.list) {
    Rsq <- which.max(sapply(mdl.list, function(mdl) {
        summary(mdl)$r.sq
    }))
    Rsq.adj <- which.max(sapply(mdl.list, function(mdl) {
        summary(mdl)$adj.r.sq
    }))
    Cp <- which.min(Cp.lm(mdl.list))
    LOOCV <- which.min(sapply(mdl.list, cv.lm))
    AIC <- which.min(sapply(mdl.list, AIC))
    BIC <- which.min(sapply(mdl.list, BIC))
    choices <- c(Rsq = Rsq, Rsq.adj = Rsq.adj, Cp = Cp, LOOCV = LOOCV, AIC = AIC,
        BIC = BIC)
    return(choices)
}
```

To put this all together, let's see what gets picked if we simulate 20 data points from the quadratic, and allow models of up to order 10:

```
selectors(poly.fit(sim.poly(n = 20, degree = 2), max.degree = 10))
##     Rsq Rsq.adj      Cp   LOOCV     AIC     BIC
##      10       2       1       2       1       1
```

Of course, one run doesn't mean much, so let's do this a bunch of times:

```
summary(t(replicate(1000, selectors(poly.fit(sim.poly(n = 20, degree = 2), max.degree = 10)))))
##       Rsq        Rsq.adj             Cp             LOOCV
##  Min.   :10   Min.   : 1.000   Min.   : 1.000   Min.   : 1.000
##  1st Qu.:10   1st Qu.: 2.000   1st Qu.: 1.000   1st Qu.: 1.000
##  Median :10   Median : 6.000   Median : 1.000   Median : 1.000
##  Mean   :10   Mean   : 5.401   Mean   : 2.582   Mean   : 1.939
##  3rd Qu.:10   3rd Qu.: 9.000   3rd Qu.: 3.000   3rd Qu.: 2.000
##  Max.   :10   Max.   :10.000   Max.   :10.000   Max.   :10.000
##       AIC             BIC
##  Min.   : 1.00   Min.   : 1.000
##  1st Qu.: 1.00   1st Qu.: 1.000
##  Median : 2.00   Median : 1.000
##  Mean   : 3.78   Mean   : 1.948
##  3rd Qu.: 7.00   3rd Qu.: 2.000
##  Max.   :10.00   Max.   :10.000
```

This is showing us the summary statistics for the degree of the polynomial model selected according to each criteria. (Why do I put in the transpose?) Remember that the right degree here is 2, so R^2 is (as usual) useless, and adjusted R^2 little better. The others all at least do something roughly right, though AIC is worse than the other three.

Of course, $n = 20$ isn't very much information[9], so let's increase that to $n = 1000$.

```
summary(t(replicate(1000, selectors(poly.fit(sim.poly(n = 1000, degree = 2),
    max.degree = 10)))))
##       Rsq        Rsq.adj             Cp             LOOCV
##  Min.   :10   Min.   : 1.000   Min.   : 1.000   Min.   : 1.00
##  1st Qu.:10   1st Qu.: 1.000   1st Qu.: 1.000   1st Qu.: 1.00
##  Median :10   Median : 4.000   Median : 1.000   Median : 1.00
##  Mean   :10   Mean   : 4.711   Mean   : 1.747   Mean   : 1.75
##  3rd Qu.:10   3rd Qu.: 8.000   3rd Qu.: 2.000   3rd Qu.: 2.00
##  Max.   :10   Max.   :10.000   Max.   :10.000   Max.   :10.00
##       AIC              BIC
##  Min.   : 1.000   Min.   :1.000
##  1st Qu.: 1.000   1st Qu.:1.000
##  Median : 1.000   Median :1.000
##  Mean   : 1.748   Mean   :1.001
##  3rd Qu.: 2.000   3rd Qu.:1.000
##  Max.   :10.000   Max.   :2.000
```

Adjusted R^2 is hopeless, leave-one-out does essentially the same as AIC or C_p (and all have a 25% probability of picking a model which is too big), and BIC is, as expected, much more conservative.

You can experiment with seeing what happens if you change the true order of the model, or the range of orders compared by the model selectors, or make some of the higher-order polynomial terms close to but not quite zero, etc.

[9] Though it seems to be enough for leave-one-out or BIC.