Shravan Vasishth, Daniel Schad, Audrey Bürki, Reinhold Kliegl

Linear Mixed Models in Linguistics and Psychology: A Comprehensive Introduction

Dedicated to ...

Contents

Preface

This book (once completed! :) is intended to be a relatively complete introduction to the application of linear mixed models in areas related to linguistics and psychology; throughout, we use the programming language R. Our target audience is cognitive scientists (e.g., linguists and psychologists) who carry out behavioral experiments, and who are interested in learning the foundational ideas behind modern statistical methodology from the ground up and in a principled manner.

Many excellent introductory textbooks already exist that discuss data analysis in great detail. Our book is different from existing books in two respects. First, our main focus is on showing how to analyze data from planned experiments involving repeated measures; this type of experimental data involves complexities that are distinct from the problems one encounters when analyzing observational data. We aim to provide many examples, with different types of dependent measures collected in a variety of experimental paradigms, including eyetracking data (visual world and reading experiments), response time data (e.g., self-paced reading, picture naming), event-related potential data, ratings (e.g., acceptability ratings), yes/no responses (e.g., speeded grammaticality judgements), and accuracy data. Second, from the very outset, we stress a particular workflow that has as its centerpiece data simulation; we aim to teach a philosophy that involves thinking about the assumed underlying generative process, *even before the data are collected*. By the "generative process", we mean the underlying assumptions about the "process" that produced the data; what exactly this means will presently become clear. The data analysis approach that we hope to teach through this book (once this book

is in its final form) involves a cycle of experiment design analysis and model validation using simulated data.

0.1 Prerequisites

This book assumes high school arithmetic and algebra. Elementary concepts of probability theory are assumed; the sum and the product rules, and the fact that the probabilities of all possible events must sum to 1, are the extent of the knowledge assumed. We also expect that the reader already knows basic constructs in the programming language R (R Core Team, 2019), such as writing for-loops. For newcomers to R, we will eventually provide a quick introduction in the appendix that covers all the constructs used in the book (this has not yet been done). For those lacking background in R, there are many good online resources on R that they can consult as needed. Examples are: R for data science[1], and Efficient R programming[2]. We also assume that the reader has done some data analysis previously, either in linguistics or psychology. This should not be the first statistics-related book they read. The reader should have encountered concepts like parameters and parameter estimation.

provide comprehensive book recommendations

0.2 How to read this book

The chapters in this book are intended to be read in sequence.

to-do: add a Mackay type chapter ordering for different scenarios.

[1] https://r4ds.had.co.nz/
[2] https://csgillespie.github.io/efficientR/

0.3 Online materials

The entire book, including all data and source code, is available online from https://github.com/vasishth/Freq_CogSci. Solutions to the exercises will be provided (to-do).

0.4 Software needed

Before you start, please install

- R[3] (and RStudio[4] or any other IDE)
- The R packages `MASS`, `dplyr`, `purrr`, `readr`, `extraDistr`, `ggplot2`, `bivariate`,`intoo`,`barsurf`, `SIN`, `kableExtra`, `tidyverse`,`hypr`,`designr`
  - These can be installed in the usual way: `install.packages(c("MASS", "dplyr", "purrr", "readr", "extraDistr", "ggplot2","bivariate","intoo","barsurf","SIN","kableExtra","tidyverse"`

In every R session, we'll need to set a seed (this ensures that the random numbers are always the same).

0.5 Acknowledgements

We are grateful to the many generations of students at the University of Potsdam, various summer schools at ESSLLI, the LOT winter school, other short courses we have taught at various institutions, and the annual summer school on Statistical Methods for Linguistics and Psychology (SMLP) at the University of Potsdam. The participants in these courses helped us considerably in improv-

[3] https://cran.r-project.org/
[4] https://www.rstudio.com/

ing the material presented here. We are also grateful to members of Vasishth lab for comments on earlier drafts of this book.

Shravan Vasishth, Daniel Schad, Audrey Bürki, Reinhold Kliegl,
Potsdam, Germany

About the Authors

Shravan Vasishth (http://vasishth.github.io) is professor of Psycholinguistics at the University of Potsdam, Germany. He holds the chair for Psycholinguistics and Neurolinguistics (Language Processing). After completing his Bachelor's degree in Japanese from Jawaharlal Nehru University, New Delhi, India, he spent five years in Osaka, Japan, studying Japanese and then working as a translator in a patent law firm in Osaka. He completed an MS in Computer and Information Science (2000-2002) and a PhD in Linguistics (1997-2002) from the Ohio State University, Columbus, USA, and an MSc in Statistics (2011-2015) from the School of Mathematics and Statistics, University of Sheffield, UK. He is a chartered statistician with the Royal Statistical Society (ID: 128307), and a member of the International Society for Bayesian Analysis. His research focuses on computational modeling of sentence processing in unimpaired and impaired populations, and the application of mathematical, computational, experimental, and statistical methods (particularly Bayesian methods) in linguistics and psychology. He runs an annual summer school, Statistical Methods in Linguistics and Psychology (SMLP): vasishth.github.io/smlp. He regularly teaches short courses on statistical data analysis (Bayesian and frequentist methods).

Daniel J. Schad (https://danielschad.github.io/) is a professor in the department of Psychology at the Health and Medical University Potsdam, Germany. He studied Psychology at the University of Potsdam, Germany, and at the University of Michigan, Ann Arbor, USA. He did a PhD in Cognitive Psychology at the University of Potsdam, working on computational models of eye-movement control and on mindless reading. He then did a five-year post-doc in the novel field of Computational Psychiatry at the

Charité, Universität Berlin, Germany (partly also at the University of Potsdam), with research visits at the ETH Zürich, Switzerland, and the University College London, UK, working on model-free and model-based decision-making and Pavlovian-instrumental transfer in alcohol dependence, and on the cognitive and brain mechanisms underlying Pavlovian conditioning. He has worked as a postdoctoral researcher at the University of Potsdam, conducting research on quantitative methods in Cognitive Science, including contrasts, properties of significance tests, Bayesian Workflow, and Bayes factor analyses.

Audrey Bürki (https://audreyburki.github.io/Website/) leads a research group at the University of Potsdam, Germany. She holds a Diploma in Speech Pathology from the University of Neuchâtel (Switzerland), a MA in Phonetic Sciences from the University of Paris 3 (France) and a PhD in linguistics from the University of Geneva (Switzerland). After her PhD she worked as a post-doc at the Universities of York (UK) and Aix-Marseille (France), and as a lecturer in Methodology and Applied statistics at the University of Geneva. Her research combines a theoretical interest for the cognitive architecture of the language production system and a methodological interest for the paradigms and statistical tools that allow collecting and analyzing the relevant data. Her research recruits a variety of methods, including phonetic analyses, corpus analyses, response-time experiments, eye-tracking and Event-Related Potentials.

Reinhold Kliegl (https://www.uni-potsdam.de/de/trainingswissenschaft/mitarbeiter/rkliegl.html) is a senior professor in the Division of Training and Movement Science, at the University of Potsdam, Germany. His research interests are… to-do

Part I

Foundational ideas

1

Some important facts about distributions

In linguistics and psychology, typical data-sets involve either *discrete* dependent measures such as acceptability ratings on a Likert scale (for example, ranging from 1 to 7), and binary grammaticality judgements, or *continuous* dependent measures such as reading times or reaction times in milliseconds and EEG signals in microvolts.

Whenever we fit a model using one of these types of dependent measures, we make some assumptions about how these measurements were generated. In particular, we usually assume that our observed measurements are coming from a particular *distribution*. The normal distribution is an example that may be familiar to the reader.

In this chapter, we will learn how to make explicit the assumptions about the distribution associated with our data; we will also learn to visualize distributions. In order to do this, we need to understand the concept of a random variable, which presupposes some basic knowledge of probability theory (such as the sum and product rules). As will become apparent in this chapter, it is extremely useful to be able to think about data in terms of the underlying random variable producing the data. We consider the two cases, discrete and continuous, separately.

We will explain the terms *random variable* and *distribution* below through examples. But it is useful to define the notion of random variable formally.

A random variable, which will be denoted by a variable such as Y, is defined as a function from a sample space of possible outcomes S to the real number system:

$$Y : S \to \mathbb{R} \tag{1.1}$$

The random variable associates to each outcome ω in the sample space S ($\omega \in S$) exactly one number $Y(\omega) = y$. S_Y will represent a set that contains all the y's (all the possible values of Y, which we call the support of Y). We can compactly write: $y \in S_Y$.

Every random variable Y has associated with it a probability mass (density) function (PMF, PDF). The term PMF is used for discrete distributions, and PDF for continuous distributions. One distinction between the PMF and PDF is crucial: the PMF maps every element of S_Y to a value between 0 and 1, whereas the PDF maps a range of values $r \in S_Y$ to a value between 0 and 1 (examples are coming up). For both PMFs and PDFs, we will express this as follows:

$$p_Y : S_Y \to [0, 1] \tag{1.2}$$

Probability mass functions (discrete case) and probability density functions (continuous case) are functions that assign probabilities (discrete case) to discrete events (discrete case) or a continuous range of values (continuous case) in a sample space.

The meanings of the terms PMF, PDF, etc., will become clearer as we discuss examples below. The reader to should revisit the above definition themselves when we present some concrete examples of discrete and continuous random variables below.

1.1 Discrete random variables: An example using the Binomial distribution

Imagine that our data come from a grammaticality judgement task (participants see or hear sentences and have to decide whether these are grammatical or ungrammatical), and that the responses from participants are a sequence of 1's and 0's, where 1 represents

the judgment "grammatical", and 0 represents the judgement "ungrammatical". Assume also that each response, coded as 1 or 0, is generated independently from the others. We can simulate the outcome of such an experiment (i.e., a sequence of 1s and 0s) in R. Let's generate the outcome of 20 such experiments with a sample size of 10 (i.e., each experiment provides us with 10 responses). For each experiment, we count the number of 1s (or number of "successes"). The R code that generated this output will be explained soon.

```
##  [1] 7 7 4 7 6 5 6 3 6 6 5 6 7 4 5 7 8 3 5 5
```

Outcomes such as this one (i.e., number of successes for a variable with two possible outcomes) follow a probability distribution p(Y). In more technical terms, we can say that the number of successes in each of the 20 simulated experiments above is being generated by a *discrete random variable* Y which has associated with it a probability distribution $p(Y)$ called the **Binomial distribution**.[1]

As mentioned above, for a discrete random variable, the probability distribution $p(Y)$ is called a **probability mass function** (PMF). The PMF defines the probability of each possible outcome. In the above example, with $n = 10$ trials, there are 11 possible outcomes: $0, \ldots, 10$ successes. Which of these outcomes is most probable depends on a numerical value called a *parameter* in the Binomial distribution that represents the probability of success. We will call this parameter θ; because it represents a probability, it must have a value between 0 and 1. The left-hand side plot in Figure 1.1 shows an example of a Binomial PMF with 10 trials and the parameter θ fixed at value 0.5. Setting θ to 0.5 leads to a PMF where the most probable outcome is 5 successes out of 10. If we had set θ to, say 0.1, then the most probable outcome would be 1 success out of 10; and if we had set θ to 0.9, then the most probable outcome would be 9 successes out of 10.

As we will see later, when we analyze data, a primary goal will be to compute a so-called *estimate* of the parameter (here, θ). In real-

[1] When an experiment consists of only a single trial (i.e., we can have a total number of only 0 or 1 successes), $p(Y)$ is called a **Bernoulli distribution**.

life situations, the value of the θ parameter will be unknown and unknowable; the data will allow us to compute a "guess" about the unknown value of the parameter. We will call this guess the estimate of the parameter.

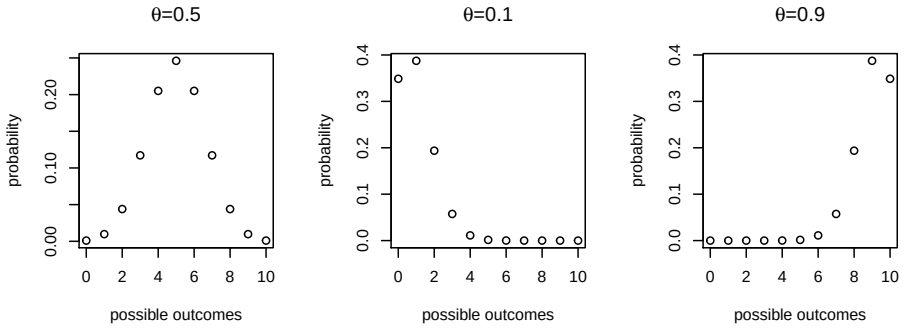


FIGURE 1.1: Probability mass functions of a binomial distribution assuming 10 trials, with 50%, 10%, and 90% probability of success.

The probability mass function for the binomial is written as follows.

$$\text{Binomial}(k|n,\theta) = \binom{n}{k}\theta^k(1-\theta)^{n-k} \tag{1.3}$$

Here, n represents the total number of trials, k the number of successes, and θ the probability of success. The term $\binom{n}{k}$, pronounced n-choose-k, represents the number of ways in which one can choose k successes out of n trials. For example, 1 success out of 10 can occur in 10 possible ways: the very first trial could be a 1, the second trial could be a 1, etc. The term $\binom{n}{k}$ expands to $\frac{n!}{k!(n-k)!}$; the exclamation mark is the factorial (e.g., $3! = 3 \times 2 \times 1$). In `R`, $\binom{n}{k}$ is computed using the function `choose(n,k)`, with n and k representing positive integer values.

1.1.1 The mean and variance of the Binomial distribution

It is possible to analytically compute the mean and variance of the PMF associated with the Binomial random variable Y. With-

out getting into the details of how these are derived mathematically, we just state here that the mean of Y (also called the expectation, conventionally written $E[Y]$) and variance of Y (written $Var(Y)$) of a Binomial distribution with parameter θ and n trials are $E[Y] = n\theta$ and $Var(Y) = n\theta(1-\theta)$.

Of course, we always know n (because we decide on the number of trials ourselves), but in real experimental situations we never know the true value of θ. But θ can be estimated from the data. From the observed data, we can compute the estimate of θ, $\hat{\theta} = k/n$. The quantity $\hat{\theta}$ is the observed proportion of successes, and is called the **maximum likelihood estimate** of the true (but unknown mean). Once we have estimated θ in this way, we can also obtain an estimate (also a maximum likelihood estimate) of the variance by computing $n\hat{\theta}(1-\hat{\theta})$. These estimates are then used for statistical inference.

What does the term "maximum likelihood estimate" mean? The term **likelihood** refers to the value of the Binomial distribution function for a particular value of θ, once we have observed some data. For example, suppose you record $n = 10$ trials, and observe $k = 7$ successes. What is the probability of observing 7 successes out of 10? We need the binomial distribution to compute this value:

$$\text{Binomial}(k = 7|n = 10, \theta) = \binom{10}{7}\theta^7(1-\theta)^{10-7} \tag{1.4}$$

Once we have observed the data, both n and k are fixed. The only variable in the above equation now is θ: the above function is now only dependent on the value of θ. When the data are fixed, the probability mass function is only dependent on the value of the parameter θ, and is called a **likelihood function**. It is therefore often expressed as a function of θ:

$p(y|\theta) = p(k = 7, n = 10|\theta) = \mathcal{L}(\theta)$

The vertical bar notation above should be read as saying that, given some data y (which in the binomial case will be k "successes" in n trials), the function returns a value for different values of θ.

If we now plot this function for all possible values of θ, we get the plot shown in Figure 1.2. We will show below how to compute this function for different values of θ.

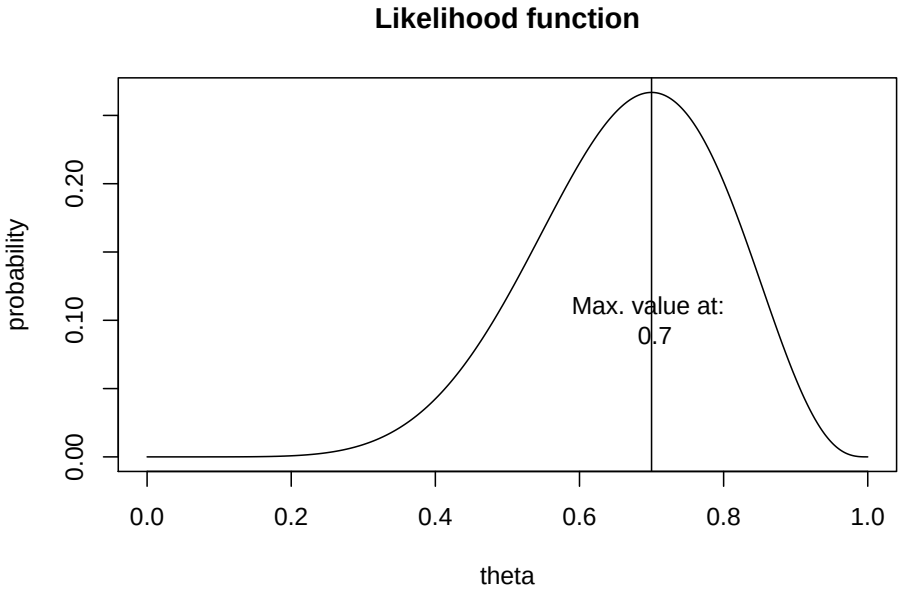


FIGURE 1.2: The likelihood function for 7 successes out of 10.

What is important about this plot is that it shows that, given the data, the maximum point is at the point 0.7, which corresponds to the estimated mean using the formula shown above: $k/n = 7/10$. Thus, the maximum likelihood estimate (MLE) gives us the most likely value of the parameter θ given the data. It is crucial to note here that the phrase "most likely" does not mean that the MLE from a *particular* sample of data invariably gives us an accurate estimate of θ. For example, if we run our experiment for 10 trials and get 1 success out of 10, the MLE is 0.10. We could have happened to observe only one success out of ten even if the true θ were 0.5. The MLE would however give an accurate estimate of the true parameter θ as n approaches infinity.

1.1.2 What information does a probability distribution provide?

What good is a probability mass function? We consider this question next. A lot of important information can be extracted from a PMF.

1.1.2.1 Compute the probability of a particular outcome (discrete case only)

The Binomial distribution shown in Figure 1.1 already displays the probability of each possible outcome under a different value for θ. In `R`, there is a built-in function that allows us to calculate $P(Y = k)$, the probability in the random variable Y of k successes out of n, given a particular value of k (this number constitutes our data), the number of trials n, and given a particular value of θ; this is the `dbinom` function. For example, the probability of 5 successes out of 10 when θ is 0.5 is:

```
(prob<-dbinom(5,size=10,prob=0.5))
```

```
## [1] 0.2461
```

To be completely explicit, we can write this probability as $P(k = 5|n = 10, \theta = 0.5) = 0.246$; when n and θ are clear from context, we can also write simply $P(k = 5)$.

The probabilities of success when θ is 0.1 or 0.7 can be computed by replacing 0.5 above by each of these probabilities:

```
dbinom(5,size=10,prob=0.1)
```

```
## [1] 0.001488
```

```
dbinom(5,size=10,prob=0.7)
```

```
## [1] 0.1029
```

One can alternatively run the above command in one shot; one can

give a range of probabilities, and get the probability of 5 successes out of 10 for these different probabilities:

```
dbinom(5,size=10,prob=c(0.1,0.7))
```

```
## [1] 0.001488 0.102919
```

The above command prints out the probability of 5 successes out of 10, when $\theta = 0.1$ and $\theta = 0.7$.

1.1.2.2 Compute the cumulative probability of k or less (more) than k successes

Instead of the probability of obtaining a given number of successes we could be interested in knowing the cumulative probability of obtaining 1 or less, or 2 or less successes. Formally, we will write this cumulative probability as $F(2)$, and more generally, as $F(k)$, for a particular value of k. Thus, $F(k) = P(Y \leq k)$.

We can compute this cumulative probability with the `dbinom` function, through a simple summation procedure:

```
## the cumulative probability of obtaining
## 0, 1, or 2 successes out of 10,
## with theta=0.5:
dbinom(0,size=10,prob=0.5)+dbinom(1,size=10,prob=0.5)+
  dbinom(2,size=10,prob=0.5)
```

```
## [1] 0.05469
```

Mathematically, we could write the above summation as:

$$\sum_{k=0}^{2} \binom{n}{k} \theta^k (1-\theta)^{n-k} \tag{1.5}$$

An alternative to the cumbersome addition in the R code above is this more compact statement, which closely mimics the above mathematical expression:

```
sum(dbinom(0:2,size=10,prob=0.5))
```

```
## [1] 0.05469
```

R has a built-in function called pbinom that does this summation for us. If we want to know the probability of 2 or less successes as in the above example, we can write:

```
pbinom(2,size=10,prob=0.5,lower.tail=TRUE)
```

```
## [1] 0.05469
```

The specification lower.tail=TRUE ensures that the summation goes from 2 to numbers smaller than 2 (which lie in the lower tail of the distribution in Figure 1.1). If we wanted to know what the probability is of obtaining 2 or more successes out of 10, we can set lower.tail to FALSE:

```
pbinom(2,size=10,prob=0.5,lower.tail=FALSE)
```

```
## [1] 0.9453
```

The cumulative distribution function or CDF, $F(k)$, can be plotted by computing the cumulative probabilities for any value k or less than k, where k ranges from 0 to 10 in our running example. The CDF is shown in Figure 1.3.

1.1.2.3 Compute the inverse of the cumulative distribution function (the quantile function)

We can also find out the value of the variable k (the quantile) such that the probability of obtaining k or less than k successes is some specific probability value p. If we switch the x and y axes of Figure 1.3, we obtain another very useful function, the inverse CDF.

The inverse of the CDF (known as the quantile function in R because it returns the quantile, the value k) is available in R as the function qbinom. The usage is as follows: to find out what the

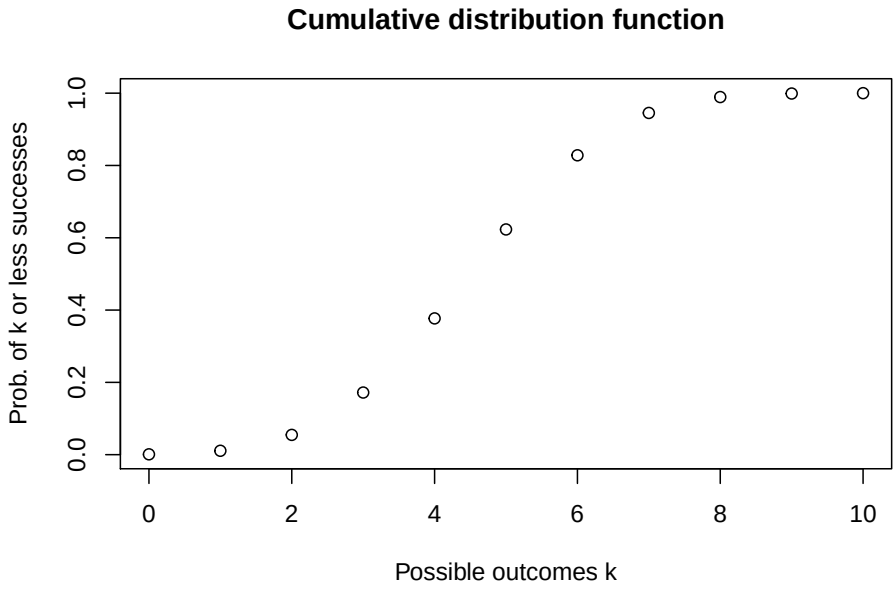


FIGURE 1.3: The cumulative distribution function for a binomial distribution assuming 10 trials, with 50% probability of success.

value k of the outcome is such that the probability of obtaining k or less successes is 0.37, type:

```
qbinom(0.37,size=10,prob=0.5)
```

```
## [1] 4
```

1.1.2.4 Generate random data from a Binomial(n, θ) distribution

We can generate random simulated data from a Binomial distribution by specifying the number of trials and the probability of success θ. In R, we do this in the following way:

```
rbinom(n=10,size=1,prob=0.5)
```

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

The above code generates a sequences of 10 randomly generated 1's

and 0's. Each of the 1's and 0's is called an outcome coming from a Bernoulli distribution. In other words, a Bernoulli distribution is just a Binomial distribution with size=1, i.e., a single trial.

Repeatedly run the above code; you will get different sequences of 1's and 0's each time. For each generated sequence, one can calculate the number of successes by just summing up the vector, or computing its mean and multiplying by the number of trials, here 10:

```
y<-rbinom(n=10,size=1,prob=0.5)
mean(y)*10 ; sum(y)
```

```
## [1] 6
```

```
## [1] 6
```

We can think of the above as 10 Bernoulli trials, or a single Binomial trial of size 10. When we see the data in this way, the syntax and output changes:

```
(y<-rbinom(n=1,size=10,prob=0.5))
```

```
## [1] 5
```

Seen as a Binomial experiment with size 10, the function returns the number of successes in a particular experiment with size 10.

Next, let's take a look at an example of a continuous random variable.

1.2 Continuous random variables: An example using the Normal distribution

We will now revisit the idea of a random variable using a continuous distribution as an example. Imagine that you have reading time data, represented as a vector y. The data y are measured in

milliseconds and are assumed to come from (i.e., assumed to be generated by) a random variable Y that has as PDF the Normal distribution (this is not a realistic assumption; we will come up with a more realistic assumption later). We will write the above statement compactly as follows:

$$Y \sim Normal(\mu, \sigma) \tag{1.6}$$

This expression should be read as: the random variable Y has PDF $Normal(\mu, \sigma)$. The random variable Y refers to an abstract mathematical object; specific instances of observed (or simulated) data will be referred to with the lower-case equivalent of the random variable; here, y can refer to a particular data-point or a vector of data (the context will make it clear whether we are talking about a single data point or a vector of data). A further important notational detail: in statistics textbooks, you will often see the Normal distribution represented in terms of the mean and the variance (σ^2) instead of the standard deviation σ that we use here. We have consciously chosen to use the standard deviation in the Normal distribution; this is because `R` uses the standard deviation and not the variance to define a normal distribution.

An important assumption we make here is that each data point in the vector of data y is independent of the others. Often, we express this assumption by saying that we have "independent and identically distributed data"; this is abbreviated as "i.i.d". One can be very explicit about this and write:

$$Y \overset{iid}{\sim} Normal(\mu, \sigma) \tag{1.7}$$

The probability density function (PDF) of the Normal distribution is defined as follows:

$$Normal(y|\mu, \sigma) = f(y) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(y-\mu)^2}{2\sigma^2}\right) \tag{1.8}$$

Here, μ is the mean, and σ is the standard deviation of the Normal distribution that the reading times have been sampled from. To

display this function, we would have to decide on specific values for μ and σ and then input the y value into the function to plot it. It may be instructive to write this function "by hand" and then plot it. We just choose some default values for μ and σ here.

```
my_normal<-function(y=NULL,mu=500,sigma=100){
  (1/sqrt(2*pi*sigma^2))*exp(-(y-mu)^2/(2*sigma^2))
}
y<-seq(100,900,by=0.01)
plot(y,my_normal(y=y),type="l")
```

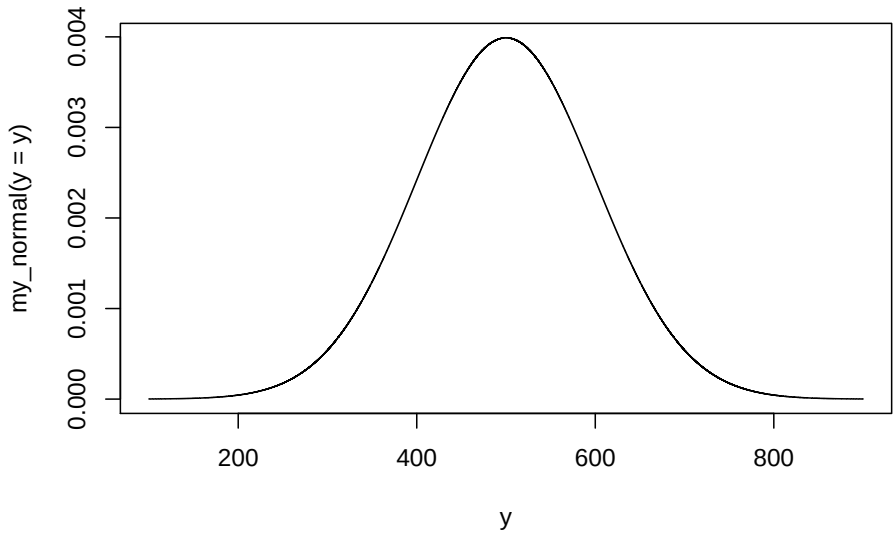


FIGURE 1.4: Creating one's own Normal distribution function.

Just like the `dbinom` function we saw earlier, there is a built-in function in `R` called `dnorm` that will allow us to display the Normal distribution. Using `dnorm`, we can visualize the Normal distribution for particular values of μ and σ as a PDF. Analogously to the discrete example we saw earlier, we can also plot the corresponding CDF (using `pnorm`), and the inverse CDF (using `qnorm`). See Figure 1.5.

In the continuous case, the PDF gives us something called the *density* for each possible value of our data y; "density" here is not

the probability, rather is it giving us a non-negative number that is the value of the function $f(y)$ in the equation immediately above. We will return to the concept of density when we do an exercise on *maximum likelihood estimation*.

As in the discrete case, the CDF tells us the probability of observing a value like y or some value less than that (written: $P(Y < y)$); and the inverse CDF gives us the quantile y such that $P(Y < y)$ is some specific value between 0 and 1. The PDF, CDF, and inverse CDF are three different ways of looking at the same information.

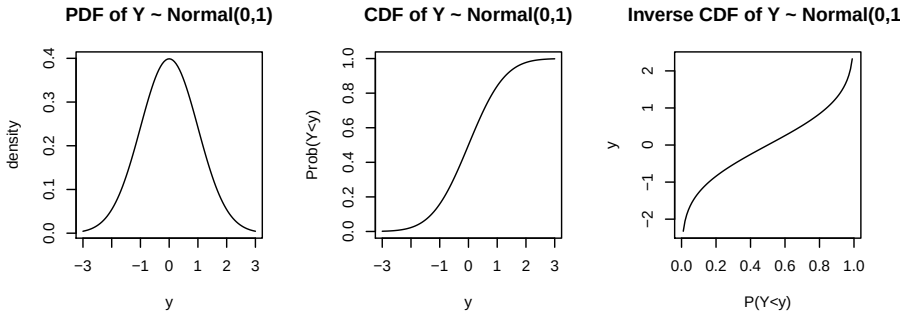


FIGURE 1.5: The PDF, CDF, and inverse CDF for the $Normal(\mu = 0, \sigma = 1)$.

One important fact about the normal distribution is that 95% of the probability mass is covered by approximately plus/minus 1.96 times the standard deviation about the mean. Thus, the range $\mu \pm 1.96 \times \sigma$ will cover approximately 95% of the area under the curve. We will approximate this by talking about $\mu \pm 2 \times \sigma$.

As in the discrete example, the PDF, CDF, and inverse of the CDF allow us to ask questions like:

- **What is the probability of observing values between some range a and b from a Normal distribution with mean μ and standard deviation σ?** We can compute the probability of the random variable lying between 1 and minus infinity:

```
pnorm(1,mean=0,sd=1)
```

```
## [1] 0.8413
```

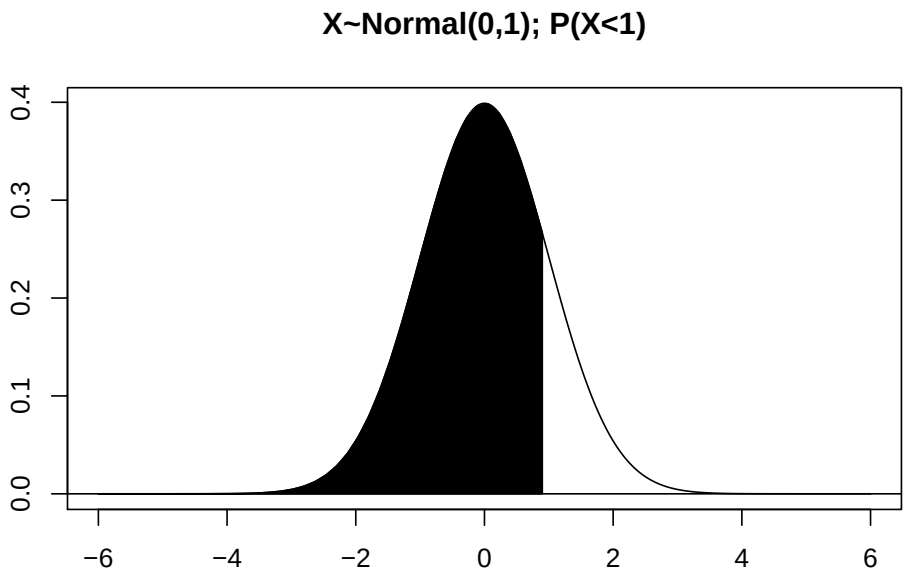


FIGURE 1.6: The area under the curve between 1 and minus infinity in a Normal distribution with mean 0 and standard deviation 1.

Notice here that the probability of any point value in a PDF is always 0. This is because the probability in a continuous probability distribution is defined to be the area under the curve, and the area under the curve at any single point on the x-axis is always 0. The implication here is that we can only ask about probabilities between a range of values; e.g., the probability that Y lies between a and b, or $P(a < Y < b)$, where $a \neq b$. Also, notice that $P(a < Y < b)$ and $P(a \leq Y \leq b)$ will be the same probability, because of the fact that $P(Y = a)$ or $P(Y = b)$ both equal 0.

- **What is the quantile q such that the probability is p of observing that value q or something less (or more) than it**? For example, we can work out the quantile q such that the probability of observing q or something less than it is

0.975, in the Normal(500,100) distribution. Formally, we would write this as $F(a) = P(Y < a)$.

```
qnorm(0.975,mean=500,sd=100)
```

```
## [1] 696
```

The above output says that the probability that the random variable is less than $q = 696$ is 97.5%.

- **Generating simulated data**. Given a vector of n independent and identically distributed data y, i.e., given that each data point is being generated independently from $Y \sim Normal(\mu, \sigma)$ for some values of the parameters μ, σ, the maximum likelihood estimates for the expectation and variance are

$$\bar{y} = \frac{\sum_{i=1}^{n} y_i}{n} \tag{1.9}$$

$$Var(y) = \frac{\sum_{i=1}^{n} (y_i - \bar{y})^2}{n} \tag{1.10}$$

As an aside, keep in mind that the formula for variance in `R` is slightly different: it divides by $n-1$ instead of n.

$$Var(y) = \frac{\sum_{i=1}^{n} (y_i - \bar{y})^2}{n-1} \tag{1.11}$$

The reason for this difference is simply that division by $n-1$ leads to a so-called unbiased estimate of the variance.

Let's simulate some sample data from a Normal distribution and compute estimates of the mean and variance.
Let's generate 10 data points using the `rnorm` function, and then compute the mean and variance from the simulated data:

```
y<-rnorm(10,mean=500,sd=100)
mean(y);var(y)
```

```
## [1] 543.1
```

```
## [1] 8037
```

As mentioned earlier, depending on the sample size, the sample mean and sample variance *from a particular sample* may or may not be close to the true values of the respective parameters, despite the fact that these are *maximum likelihood estimates*. The estimates of the mean and variance from a particular sample will be close to the true values of the parameters as the sample size goes to infinity.

1.3 Other common distributions

Here, we briefly present some of the distributions that we will need or use in this book. Throughout, we assume independent and identically distributed data.

1.3.1 The t-distribution

This continuous distribution, written $t(n-1)$, take as a parameter the degrees of freedom (roughly, the sample size n) minus one. As the degrees of freedom increase to infinity (as the sample size increases to infinity), the distribution approaches the Normal distribution with mean 0 and standard deviation 1.

The pdf is:

$$f(x) = \frac{\Gamma((n+1)/2)}{\sqrt{n\pi}\Gamma(n/2)}(1+x^2/n)^{-(n+1)/2} \text{ Support: } x \in (-\infty, \infty) \tag{1.12}$$

$\Gamma()$ is the Gamma function: $\Gamma(n) = n-1!$. The mean of the t-distribution is 0 if $n > 1$ and variance $\frac{n}{n-2}$ if $n > 2$.

The t-distribution becomes the Cauchy distribution if $n = 2$. The Cauchy distribution has no mean or variance defined for it.

There are four functions in `R` that serve the same purpose as the

`dnorm`, `pnorm`, `qnorm`, `rnorm` functions for the Normal: `dt`, `pt`, `qt`, `rt`.

1.3.2 The Gamma distribution

The distribution is written $Gamma(a, b)$, and takes two parameters, a and b.

The pdf is:

$$f(x) = \frac{1}{\Gamma(a)}(bx)^a e^{-bx}\frac{1}{x} \text{ Support: } x \in (0, \infty) \tag{1.13}$$

The mean is $\frac{a}{b}$ and the variance is $\frac{a}{b^2}$.

1.3.3 The Exponential distribution

The Exponential, written $Exp(\lambda)$, takes the parameter λ and has the PDF:

$$f(x) = \lambda e^{-\lambda x} \text{ Support: } x \in (0, \infty) \tag{1.14}$$

Its mean is $\frac{1}{\lambda}$ and variance $\frac{1}{\lambda^2}$.

1.4 Bivariate and multivariate distributions

So far, we have only discussed univariate distributions. It is also possible to specify distributions with two or more dimensions.

Understanding bivariate (and, more generally, multivariate) distributions, and knowing how to simulate data from such distributions, is vital for us because linear mixed models crucially depend on such distributions. If we want to understand linear mixed models, we have to understand multivariate distributions.

1.4.1 Example 1: Discrete bivariate distributions

Starting with the discrete case, consider the discrete bivariate distribution shown below. These are data from an experiment where, inter alia, in each trial a Likert acceptability rating and a response accuracy (to a yes-no question) were recorded (the data are from a study by Laurinavichyute (2020), used with permission here).

Figure 1.7 shows the *joint probability mass function* of two random variables X and Y. The random variable X consists of 7 possible values (this is the 1-7 Likert response scale), and the random variable Y is response accuracy, with 0 representing incorrect responses, and 1 representing correct responses.

```
agrmt<-read.csv("data/agrmt_discrete_binomial.csv")
rating0<-table(subset(agrmt,accuracy==0)$rating)
rating1<-table(subset(agrmt,accuracy==1)$rating)

ratingsbivar<-data.frame(rating0=rating0,
                         rating1=rating1)

ratingsbivar<-ratingsbivar[,c(2,4)]
colnames(ratingsbivar)<-c(0,1)
library (MASS)

## function from bivariate package:
f <- cbvpmf (ratingsbivar)
plot (f, TRUE,
       arrows=FALSE)
```

One can also display the figure as a table.

```
probs<-attr(f,"p")
t(probs)
```

```
##       [,1]    [,2]    [,3]    [,4]    [,5]    [,6]
## 0 0.01792 0.02328 0.04004 0.04306 0.06331 0.04888
```

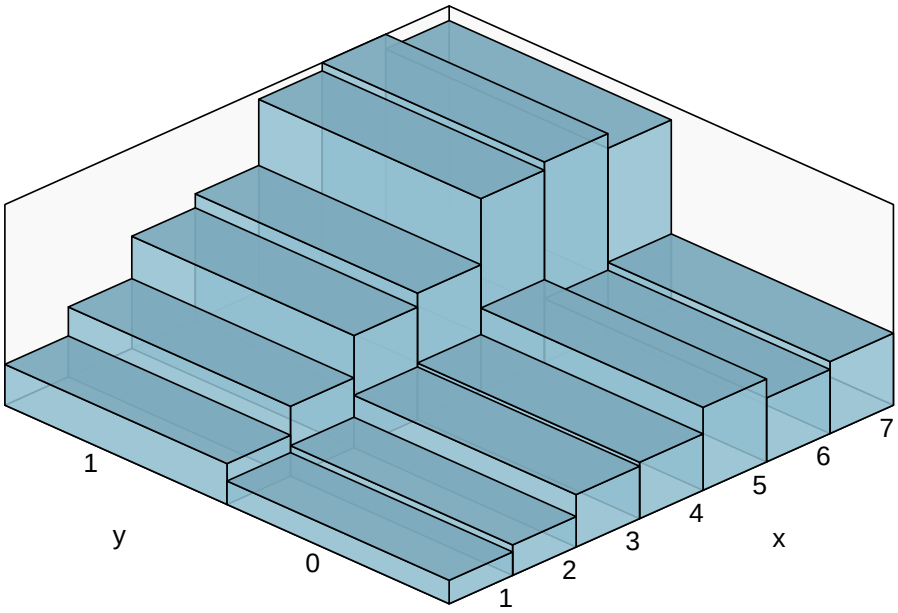


FIGURE 1.7: Example of a discrete bivariate distribution. In these data, in every trial, two pieces of information were collected: Likert responses and yes-no question responses. The random variable X represents Likert scale responses on a scale of 1-7. and the random variable Y represents 0, 1 (incorrect, correct) responses to comprehension questions.

```
## 1 0.03119 0.05331 0.08566 0.09637 0.14688 0.15317
##       [,7]
## 0 0.05493
## 1 0.14199
```

For each possible value of X and Y, we have a joint probability. Given such a bivariate distribution, there are two useful quantities we can compute: the *marginal* distributions (p_X and p_Y), and the *conditional* distributions ($p_{X|Y}$ and $p_{Y|X}$).
The table below shows the joint probability mass function $p_{X,Y}(x, y)$.

The marginal distribution p_Y is defined as follows. S_X is the support of X, i.e., all the possible values of X.

$p_{X,Y}$	x=1	x=2	x=3	x=4	x=5	x=6	x=7
y = 0	0.018	0.023	0.040	0.043	0.063	0.049	0.055
y = 1	0.031	0.053	0.086	0.096	0.147	0.153	0.142

TABLE 1.1: The joint PMF for two random variables X and Y.

$$p_Y(y) = \sum_{x \in S_X} p_{X,Y}(x, y). \tag{1.15}$$

Similarly, the marginal distribution p_X is defined as:

$$p_X(x) = \sum_{y \in S_Y} p_{X,Y}(x, y). \tag{1.16}$$

p_Y is easily computed, by summing up the values in each row; and p_X by summing up the values in each column. You can see why this is called the marginal distribution; the result appears in the margins of the table.

```
#P(Y)
(PY<-rowSums(t(probs)))
```

```
##      0      1
## 0.2914 0.7086
```

```
sum(PY) ## sums to 1
```

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

```
#P(X)
(PX<-colSums(t(probs)))
```

```
## [1] 0.04912 0.07658 0.12570 0.13943 0.21020 0.20205
## [7] 0.19693
```

```
sum(PX) ## sums to 1
```

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

The marginal probabilities sum to 1, as they should. The table below shows the marginal probabilities.

$p_{X,Y}$	x=1	x=2	x=3	x=4	x=5	x=6	x=7	P(Y)
y = 0	0.018	0.023	0.040	0.043	0.063	0.049	0.055	**0.291**
y = 1	0.031	0.053	0.086	0.096	0.147	0.153	0.142	**0.709**
P(X)	**0.049**	**0.077**	**0.126**	**0.139**	**0.210**	**0.202**	**0.197**	

TABLE 1.2: The joint and marginal distributions of X and Y.

Notice that to compute the marginal distribution of X, one is summing over all the Ys; and to compute the marginal distribution of Y, one sums over all the X's. We say that we are *marginalizing out* the random variable that we are summing over. One can visualize the two marginal distributions using barplots.

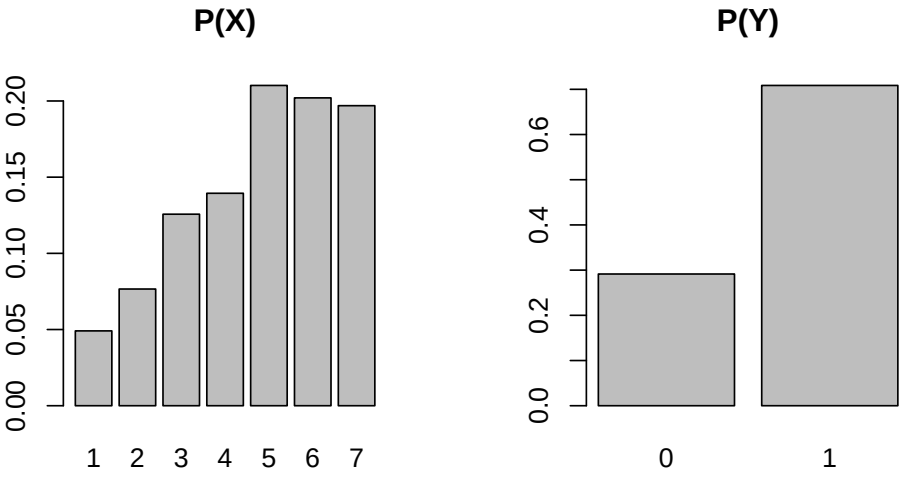


FIGURE 1.8: Marginal distributions of X and Y.

For computing conditional distributions, recall that the conditional distribution of a random variable X given that $Y = y$, where y is some specific (fixed) value, is:

$$p_{X|Y}(x \mid y) = \frac{p_{X,Y}(x,y)}{p_Y(y)} \quad \text{provided } p_Y(y) = P(Y = y) > 0 \tag{1.17}$$

As an example, let's consider how $p_{X|Y}$ would be computed. The possible values of y are $0, 1$, and so we have to find the conditional distribution (defined above) for each of these values. I.e., we have to find $p_{X|Y}(x \mid y = 0)$, and $p_{X|Y}(x \mid y = 1)$.

Let's do the calculation for $p_{X|Y}(x \mid y = 0)$.

$$\begin{aligned} p_{X|Y}(1 \mid 0) =& \frac{p_{X,Y}(1,0)}{p_Y(0)} \\ =& \frac{0.018}{0.291} \\ =& 0.0619 \end{aligned} \tag{1.18}$$

This conditional probability value will occupy the cell X=1, Y=0 in the table below summarizing the conditional probability distribution $p_{X|Y}$. In this way, one can fill in the entire table, which will then represent the conditional distributions $p_{X|Y=0}$ and $p_{X|Y=1}$. The reader may want to take a few minutes to complete the table.

	x=1	x=2	x=3	x=4	x=5	x=6	x=7
$p_{X\|Y}(x \mid y = 0)$	0.0619						
$p_{X\|Y}(x \mid y = 1)$							

TABLE 1.3: The conditional probability distribution of X given Y.

Similarly, one can construct a table that shows $p_{Y|X}$.

1.4.2 Example 2: Continuous bivariate distributions

Consider now the continuous bivariate case; this time, we will use simulated data. Consider two normal random variables X and Y,

each of which coming from, for example, a Normal(0,1) distribution, with some correlation ρ between the two random variables.

A bivariate distribution for two random variables X and Y, each of which comes from a normal distribution, is expressed in terms of the means and standard deviations of each of the two distributions, and the correlation ρ between them. The standard deviations and correlation are expressed in a special form of a 2×2 matrix called a variance-covariance matrix Σ. If ρ_u is the correlation between the two random variables, and σ_x and σ_y the respective standard deviations, the variance-covariance matrix is written as:

$$\Sigma = \begin{pmatrix} \sigma_x^2 & \rho\sigma_x\sigma_y \\ \rho\sigma_x\sigma_y & \sigma_y^2 \end{pmatrix} \tag{1.19}$$

The off-diagonals of this matrix contain the covariance between X and Y.

The joint distribution of X and Y is defined as follows:

$$\begin{pmatrix} X \\ Y \end{pmatrix} \sim \mathcal{N}_2 \left(\begin{pmatrix} 0 \\ 0 \end{pmatrix}, \Sigma \right) \tag{1.20}$$

The subscript on $\mathcal{N}_2$ refers to the number of dimensions; if we had a multivariate distribution with three random variables, say X, Y, Z, the distribution would be $\mathcal{N}_3$, and so on. The variance-covariance matrix for the three-dimensional distribution would be a 3×3 matrix, not a 2×2 matrix as above, and would contain three correlations $(\rho_{X,Y}, \rho_{X,Z}, \rho_{Y,Z})$.

Returning to the bivariate example, the joint PDF is written with reference to the two variables $f_{X,Y}(x, y)$. It has the property that the area under the curve sums to 1. Formally, we would write this as a double integral: we are summing up the area under the curve for both dimensions X and Y. The integral symbol ($\int$) is just the continuous equivalent of the discrete summation symbol ($\sum$).

$$\iint_{S_{X,Y}} f_{X,Y}(x,y)\,dxdy = 1 \tag{1.21}$$

Here, the terms dx and dy express the fact that we are summing the area under the curve along the X axis and the Y axis.

The joint CDF would be written as follows. The equation below gives us the probability of observing a value like (u, v) or some value smaller than that (i.e., some (u', v'), such that $u' < u$ and $v' < v$.

$$\begin{aligned} F_{X,Y}(u,v) =& P(X < u, Y < v) \\ =& \int_{-\infty}^{u}\int_{-\infty}^{v} f_{X,Y}(x,y)\,dydx \text{ for } (x,y) \in \mathbb{R}^2 \end{aligned} \tag{1.22}$$

As an aside, notice that the support for the normal distribution ranges from minus infinity to plus infinity. There can however be other PDFs with a more limited support; an example would be a normal distribution whose pdf $f(x)$ is such that the lower bound is truncated at, say, 0. In such a case, the area under the range $\int_{-\infty}^{0} f(x)\,dx$ will be 0 because the range lies outside the support of the truncated normal distribution.

A visualization will help. The figures below show a bivariate distribution with correlation zero (Figure 1.9), a positive (Figure 1.10) and a negative correlation (Figure 1.11).

In this book, we will make use of such multivariate distributions a lot, and it will soon become important to know how to generate simulated bivariate or multivariate data that is correlated. So let's look at how to generate simulated data next.

1.4.3 Generate simulated bivariate (multivariate) data

Suppose we want to generate 100 correlated pairs of data, with correlation $\rho = 0.6$. The two random variables have mean 0, and standard deviations 5 and 10 respectively.

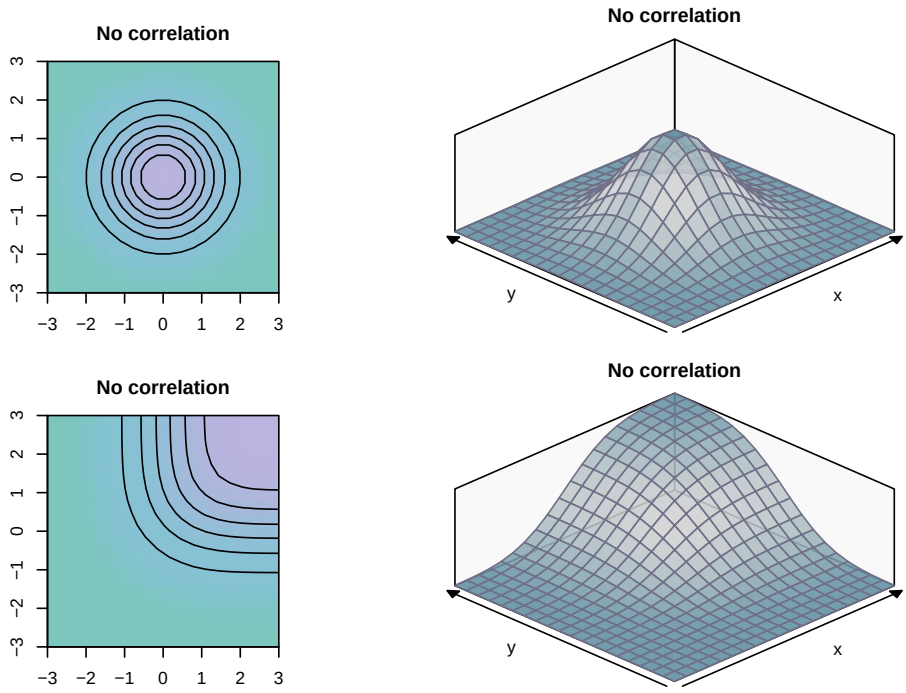


FIGURE 1.9: A bivariate Normal distribution with zero correlation. Shown are four plots: the top-right plot shows the three-dimensional bivariate density, the top-left plot the contour plot of the distribution (seen from above). The lower plots show the cumulative distribution function from two views, as a three-dimensional plot and as a contour plot.

Here is how we would generate such data. First, define a variance-covariance matrix; then, use the multivariate analog of the `rnorm` function, `mvrnorm`, to generate 100 data points.

```
library(MASS)
## define a variance-covariance matrix:
Sigma<-matrix(c(5^2,5*10*.6,5*10*.6,10^2),
              byrow=FALSE,ncol=2)
## generate data:
u<-mvrnorm(n=100,
           mu=c(0,0),
```

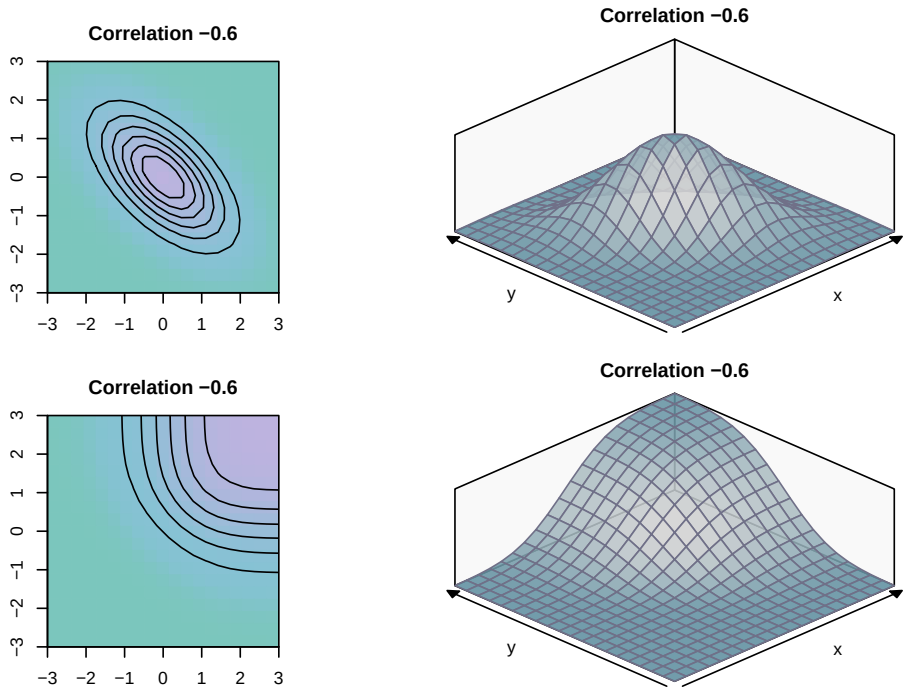


FIGURE 1.10: A bivariate Normal distribution with a positive correlation of 0.6. Shown are four plots: the top-right plot shows the three-dimensional bivariate density, the top-left plot the contour plot of the distribution (seen from above). The lower plots show the cumulative distribution function from two views, as a three-dimensional plot and as a contour plot.

```
            Sigma=Sigma)
head(u)
```

```
##          [,1]    [,2]
## [1,] 11.5225  19.530
## [2,] -5.1793   3.682
## [3,]  0.8948 -10.116
## [4,]  6.2526   8.356
## [5,] -6.7940  -5.466
## [6,] -6.5897  -9.700
```

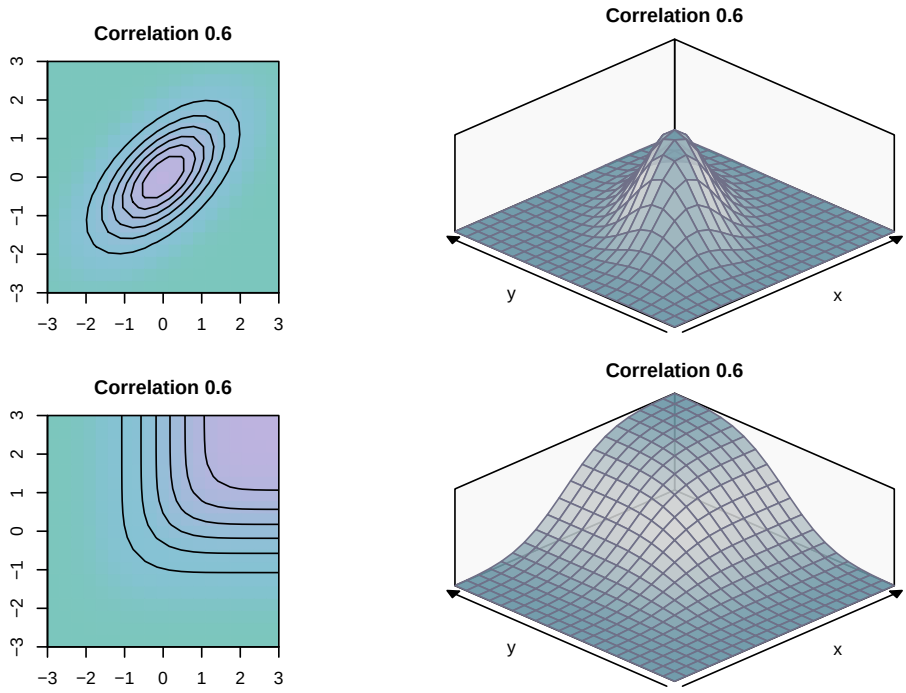
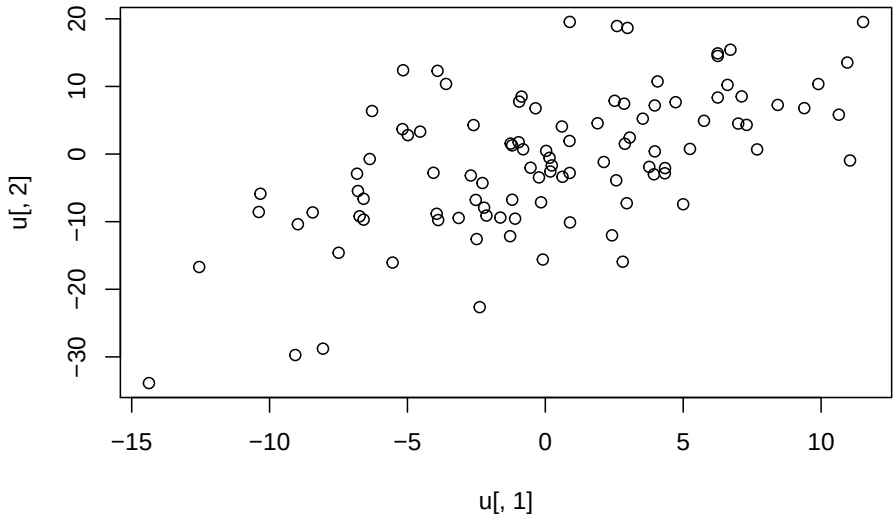


FIGURE 1.11: A bivariate Normal distribution with a negative correlation of -0.6. Shown are four plots: the top-right plot shows the three-dimensional bivariate density, the top-left plot the contour plot of the distribution (seen from above). The lower plots show the cumulative distribution function from two views, as a three-dimensional plot and as a contour plot.

A plot confirms that the simulated data are positively correlated.

```
plot(u[,1],u[,2])
```



As an exercise, try changing the correlation to 0 or to −0.6, and then plot the bivariate distribution that results.

One final useful thing to notice about the variance-covariance matrix is that it can be decomposed into the component standard deviations and an underlying correlation matrix. For example, consider the matrix above:

```
Sigma
```

```
##      [,1] [,2]
## [1,]   25   30
## [2,]   30  100
```

One can decompose the matrix as follows. The matrix can be seen as the product of a diagonal matrix of the standard deviations and the correlation matrix:

```
## sds:
(sds<-c(5,10))
```

```
## [1]  5 10
```

```
## diagonal matrix:
(sd_diag<-diag(sds))
```

```
##      [,1] [,2]
## [1,]    5    0
## [2,]    0   10
```

```
## correlation matrix:
(corrmatrix<-matrix(c(1,0.6,0.6,1),ncol=2))
```

```
##      [,1] [,2]
## [1,]  1.0  0.6
## [2,]  0.6  1.0
```

Given these two matrices, one can reassemble the variance-covariance matrix:

```
sd_diag%*%corrmatrix%*%sd_diag
```

```
##      [,1] [,2]
## [1,]   25   30
## [2,]   30  100
```

There is a built-in convenience function, `sdcor2cov` in the `SIN` package that does this calculation, taking the vector of standard deviations (not the diagonal matrix) and the correlation matrix to yield the variance-covariance matrix:

```
SIN::sdcor2cov(stddev=sds,corr=corrmatrix)
```

```
##      [,1] [,2]
## [1,]   25   30
## [2,]   30  100
```

We will be using this function a lot when simulating data from hierarchical models.

1.5 Likelihood and maximum likelihood estimation

We now turn to an important topic: the idea of likelihood, and of maximum likelihood estimation. Consider as a first example the discrete case, using the Binomial distribution.

Suppose we toss a fair coin 10 times, and count the number of heads; we do this experiment once. Notice below that we set the probability of success to be 0.5. This is because we are assuming that we tossed a fair coin.

```
(x<-rbinom(1,size=10,prob=0.5))
```

```
## [1] 3
```

The *probability* of obtaining this value depends on the parameter we set for θ in the PMF for the binomial distribution. Here are some possible values for θ and the resulting probabilities:

```
dbinom(x,size=10,prob=0.1)
```

```
## [1] 0.0574
```

```
dbinom(x,size=10,prob=0.3)
```

```
## [1] 0.2668
```

```
dbinom(x,size=10,prob=0.5)
```

```
## [1] 0.1172
```

```
dbinom(x,size=10,prob=0.8)
```

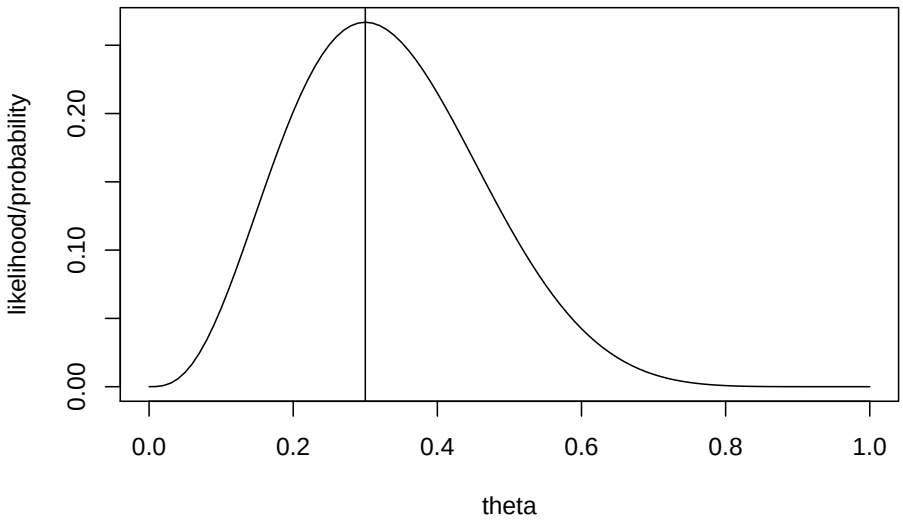
```
## [1] 0.0007864
```

```
dbinom(x,size=10,prob=1.0)
```

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

The value of θ that gives us the highest probability will be called the **maximum likelihood estimate**. The function `dbinom` (which is a function of θ) is also called a likelihood function, and the maximum value of this function is called the maximum likelihood estimate. We can graphically figure out the maximal value of the `dbinom` likelihood function here by plotting the value of the function for all possible values of θ, and checking which is the maximal value:

```
theta<-seq(0,1,by=0.01)
plot(theta,dbinom(x,size=10,prob=theta),type="l",
     ylab="likelihood/probability")
abline(v=x/10)
```



It should be clear from the figure that the maximum value corresponds to the proportion of heads: 3/10. This value is called the maximum likelihood estimate (MLE). We can obtain this MLE of θ, which maximizes the likelihood, by computing:

$$\hat{\theta} = \frac{x}{n} \tag{1.23}$$

where n is sample size, and x is the number of successes. For the analytical derivation of this result, see the Linear Modeling lecture notes: https://github.com/vasishth/LM

The likelihood function in a continuous case is similar to that of the discrete example above, but there is one crucial difference, which we will just get to below.

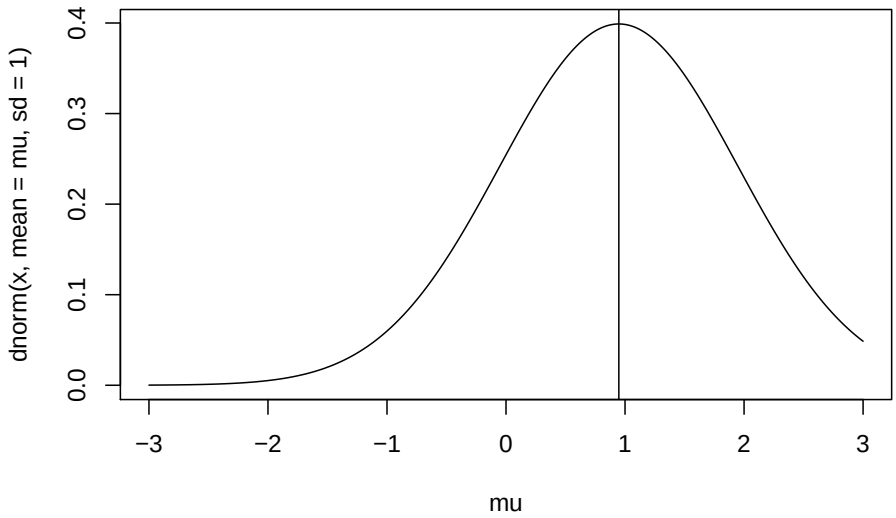
Consider the following example. Suppose we have one data point from a Normal distribution, with mean 0 and standard deviation 1:

```
(x<-rnorm(1,mean=0,sd=1))
```

```
## [1] 0.948
```

The likelihood function for this data point is going to depend on two parameters, μ and σ. For simplicity, let's assume that σ is known to be 1 and that only μ is unknown. In this situation, the value of μ that maximizes the likelihood will be the MLE. As before, we can graphically find the MLE by plotting the likelihood function:

```
mu<-seq(-3,3,by=0.01)
plot(mu,dnorm(x,mean=mu,
              sd=1),type="l")
abline(v=x)
```



The maximum point in this function will always be the sample mean from the data; the sample mean is the MLE. In the above case, the mean of the single data point 0.948 is the number itself. If we had two data points from a Normal(0,1) distribution, then the likelihood function would be defined as follows. First, let us simulate two data points:

```
(x1<-rnorm(1))
```

```
## [1] -1.202
```

```
(x2<-rnorm(1))
```

```
## [1] -0.4661
```

These two data points are independent of each other. Hence, to obtain the joint likelihood, we will have to multiply the likelihoods of each of the numbers, given some value for μ:

```
mu<-1
dnorm(x1,mean=mu)*dnorm(x2,mean=mu)
```

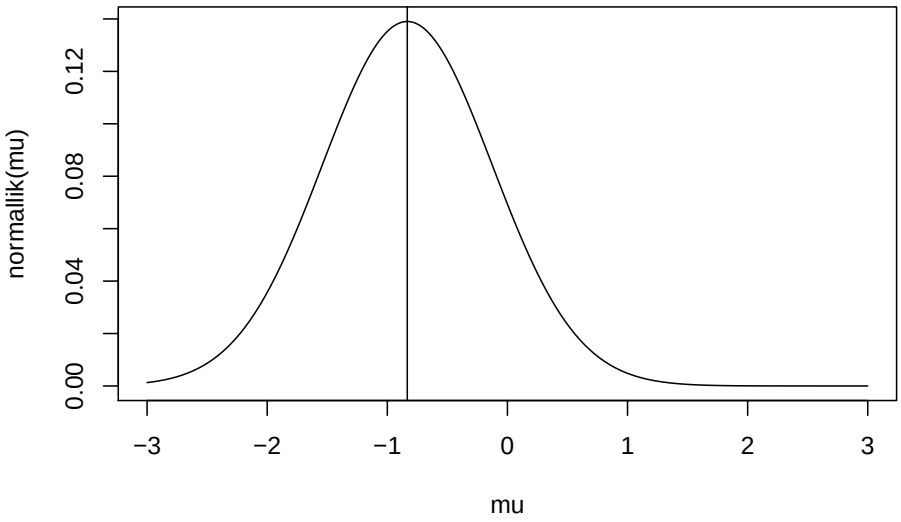
In order to plot the joint likelihood, we need to write a function:

```
normallik<-function(mu=NULL){
  dnorm(x1,mean=mu)*dnorm(x2,mean=mu)
}
## test:
normallik(mu=1)
```

```
## [1] 0.004815
```

Now, we can plot the likelihood function for these two data points:

```
mu<-seq(-3,3,by=0.01)
plot(mu,normallik(mu),type="l")
abline(v=mean(c(x1,x2)))
```



Notice that the maximum value of this joint likelihood is the mean of the two data points.

For the normal distribution, where $Y \sim N(\mu, \sigma)$, we can get MLEs of μ and σ by computing:

$$\hat{\mu} = \frac{1}{n} \sum y_i = \bar{y} \tag{1.24}$$

and

$$\hat{\sigma}^2 = \frac{1}{n}\sum(y_i - \bar{y})^2 \tag{1.25}$$

As mentioned earlier, as the formula for the variance, you will sometimes see the unbiased estimate (and this is what R computes) but for large sample sizes the difference is not important:

$$\hat{\sigma}^2 = \frac{1}{n-1}\sum(y_i - \bar{y})^2 \tag{1.26}$$

Now we come to a crucial difference between the discrete and continuous cases discussed above. The `dbinom` is the PMF, but it is also a likelihood function when seen as a function of θ. Once we have fixed the θ parameter to a particular value, the `dbinom` function gives us the *probability* of a particular outcome. Given some data k from n trials from a Binomial distribution, and treating θ as variable between 0 and 1, `dbinom` gives us the likelihood.

The situation is slightly different in the continuous PDF. Taking the normal distribution as an example, the `dnorm` function $f(y|\mu, \sigma)$ doesn't give us the **probability** of a point value, but rather the **density**. When the function $f(y|\mu, \sigma)$ is treated as a function of the parameters, it gives us the **likelihood**. We need to make a careful distinction between the words probability and likelihood; in day-to-day usage the two words are used interchangeably, but here these two terms have different technical meanings.

A final point to note is that a likelihood function is not a PDF; the area under the curve does not need to sum to 1. By contrast, in a PDF, the area under the curve must sum to 1.

1.5.1 The importance of the MLE

One significance of the MLE is that, having assumed a particular underlying PMF/PDF, we can estimate the (unknown) parameters (the mean and variance) of the distribution that we assume to have generated our particular data. For example, in the Binomial

case, we have a formula for computing the MLEs of the mean and variance; for the Normal distribution, we have a formula for computing the MLE of the mean and the variance.

What are the distributional properties of the mean **under repeated sampling**? This is a question that forms the basis for hypothesis testing in the frequentist paradigm. So we turn to this topic in the next chapter.

1.6 Summary of useful R functions relating to univariate distributions

Table 1.4 summarizes the different functions relating to univariate PMFs and PDFs, using the Binomial and Normal as examples.

TABLE 1.4: Important R functions relating to two univariate distributions, the Binomial and the Normal. In the table, prob represents probability and ranges from 0 to 1. $P(y)$ is the probability of observing y; $F(y)$ is the cumulative distribution function (CDF); and $F^{-1}(prob)$ is the inverse of the CDF.

	Discrete	Continuous
Example:	Binomial(n,θ)	Normal(μ, σ)
Likelihood function	dbinom	dnorm
Probability: $P(Y = y)$	dbinom	always 0
CDF, $F(y) = P(Y \geq y) = prob$	pbinom	pnorm
Inverse CDF, $F^{-1}(prob) = y$	qbinom	qnorm
Generate simulated data	rbinom	rnorm

Other distributions, such as the t-distribution, the Uniform, Exponential, Gamma, Beta, etc., have their own set of d-p-q-r functions in R.

1.7 Summary of random variable theory

We can summarize the above informal concepts relating to random variables very compactly if we re-state them in mathematical form. A mathematical statement has the advantage not only of brevity but also of reducing ambiguity.

Formally, a random variable Y is defined as a function from a sample space of possible outcomes S to the real number system:

$$Y : S \to \mathbb{R} \tag{1.27}$$

The random variable associates to each outcome $\omega \in S$ exactly one number $Y(\omega) = y$. S_Y is all the y's (all the possible values of Y, the support of Y). I.e., $y \in S_Y$.

Every random variable Y has associated with it a probability mass (distribution) function (PMF, PDF). I.e., PMF is used for discrete distributions and PDF for continuous distributions. The PMF maps every element of S_Y to a value between 0 and 1. The PDF maps a range of values in the support of Y to a value between 0 and 1 (e.g., $P(a \leq Y \leq b) \to [0, 1]$).

$$p_Y : S_Y \to [0, 1] \tag{1.28}$$

Probability mass functions (discrete case) and probability density functions (continuous case) are functions that assign probabilities or relative frequencies to events in a sample space.

The expression

$$Y \sim f(\cdot) \tag{1.29}$$

will be used to mean that the random variable Y has PDF/PMF $f(\cdot)$. For example, if we say that $Y \sim Binomial(n, \theta)$, then we are asserting that the PMF is:

$$\text{Binomial}(k|n,\theta) = \binom{n}{k}\theta^k(1-\theta)^{n-k} \tag{1.30}$$

If we say that $Y \sim Normal(\mu, \sigma)$, we are asserting that the PDF is

$$Normal(y|\mu,\sigma) = \frac{1}{\sqrt{2\pi\sigma^2}}\exp\left(-\frac{y-\mu)^2}{2\sigma^2}\right) \tag{1.31}$$

The **cumulative distribution function** or CDF is defined as follows:

For discrete distributions, the probability that Y is less than a is written:

$$P(Y < a) = F(Y < a) = \sum_{-\infty}^{a} f(y) \tag{1.32}$$

For continuous distributions, the summation symbol $\sum$ above becomes the summation symbol for the continuous case, which is the integral $\int$. The upper and lower bounds are marked by adding a subscript and a superscript on the integral. For example, if we want the area under the curve between points a and b for some function $f(y)$, we write $\int_b^a f(y)\,dy$. So, if we want the probability that Y is less than a, we would write:

$$P(Y < a) = F(Y < a) = \int_{-\infty}^{a} f(y)\,dy \tag{1.33}$$

The above integral is simply summing up the area under the curve between the points $-\infty$ and a; this gives us the probability of observing a or a value smaller than a.

We can use the complementary cumulative distribution function to compute quantities like $P(Y > a)$ by computing $1 - F(a)$, and the quantity $P(a \leq Y \leq b)$ by computing $F(b) - F(a)$, where $b > a$.

A final point here is that we can go back and forth between the PDF and the CDF. If the PDF is $f(y)$, then the CDF that allows us to compute quantities like $P(Y < b)$ is just the integral:

$$F(Y < b) = \int_{-\infty}^{b} f(y)\, dy \tag{1.34}$$

The above is simply computing the area under the curve $f(y)$, ranging from b to $-\infty$.

Because differentiation is the opposite of integration (this is called the Fundamental Theorem of Calculus), if we differentiate the CDF, we get the PDF back:

$$d(F(y))/dy = f(y) \tag{1.35}$$

In bivariate distributions, the joint CDF is written $F_{X,Y}(a, b) = P(X \leq a, Y \leq b)$, where $-\infty < a, b < \infty$. The *marginal distributions* of F_X and F_Y are the CDFs of each of the associated random variables. The CDF of X:

$$F_X(a) = P(X \leq a) = F_X(a, \infty) \tag{1.36}$$

The CDF of Y:

$$F_Y(b) = P(Y \leq b) = F_Y(\infty, b) \tag{1.37}$$

$f(x, y)$ is the *joint PDF* of X and Y. Every joint PDF satisfies

$$f(x, y) \geq 0 \text{ for all } (x, y) \in S_{X,Y}, \tag{1.38}$$

and

$$\int\int_{S_{X,Y}} f(x, y)\, \mathrm{d}x\, \mathrm{d}y = 1. \tag{1.39}$$

where $S_{X,Y}$ is the joint support of the two random variables.

If X and Y are jointly continuous, they are individually continuous, and their PDFs are:

$$\begin{aligned} P(X \in A) =& P(X \in A, Y \in (-\infty, \infty)) \\ =& \int_A \int_{-\infty}^{\infty} f(x,y)\, dy\, dx \\ =& \int_A f_X(x)\, dx \end{aligned} \tag{1.40}$$

where

$$f_X(x) = \int_{-\infty}^{\infty} f(x,y)\, dy \tag{1.41}$$

Similarly:

$$f_Y(y) = \int_{-\infty}^{\infty} f(x,y)\, dx \tag{1.42}$$

1.8 Further reading

For readers interested in the mathematics needed for statistics, the books by Fox (2009), Gill (2006), and Moore and Siegel (2013) are useful. The essential matrix algebra needed for statistics is discussed in Fieller (2016). Accessible introductions to probability theory are Morin (2016) and Blitzstein and Hwang (2014). Kerns (2010) contains a very well-written and freely available general introduction to random variable theory and statistics, but assumes the reader knows the basics of calculus.

1.9 Exercises

1.9.1 Practice using the `pnorm` function

1.9.1.1 Part 1

Given a normal distribution with mean `57` and standard deviation `100`, use the pnorm function to calculate the probability of obtaining values between `222` and `84` from this distribution.

1.9.1.2 Part 2

Calculate the following probabilities. Given a normal distribution with mean `51` and standard deviation `4`, what is the probability of getting

- a score of `41` or less
- a score of `41` or more
- a score of `54` or more

1.9.1.3 Part 3

Given a normal distribution with mean `51` and standard deviation `7`, what is the probability of getting

- a score of `46` or less.
- a score between `48` and `54`.
- a score of `mu+1` or more.

1.9.2 Practice using the `qnorm` function

1.9.2.1 Part 1

Consider a normal distribution with mean 1 and standard deviation 1.

Compute the lower and upper boundaries such that:

- the area (the probability) to the left of the lower boundary is `0.37`.
- the area (the probability) to the left of the upper boundary is `0.94`.

1.9.2.2 Part 2

Given a normal distribution with mean `50.643` and standard deviation `0.736`. There exist two quantiles, the lower quantile q1 and the upper quantile q2, that are equidistant from the mean `50.643`, such that the area under the curve of the Normal probability between q1 and q2 is `85`%. Find q1 and q2.

1.9.3 Maximum likelihood estimation 1

The function `dnorm` gives the likelihood given a data point (or multiple data) and a value for the mean and the standard deviation (sd). Using `dnorm`, compute

- the likelihood of the data point `11.664` assuming a mean of `12` and standard deviation 5.
- the likelihood of the data point `11.664` assuming a mean of `11` and standard deviation 5.
- the likelihood of the data point `11.664` assuming a mean of `10` and standard deviation 5.
- the likelihood of the data point `11.664` assuming a mean of `9` and standard deviation 5.

1.9.4 Maximum likelihood estimation 2

You are given 10 independent and identically distributed data points that are assumed to come from a Normal distribution with unknown mean and unknown standard deviation:

```
x
```

```
##  [1] 503 487 511 488 516 501 488 484 493 505
```

The function `dnorm` gives the likelihood given multiple data points and a value for the mean and the standard deviation. The log-likelihood can be computed by typing `dnorm(...,log=TRUE)`.

The product of the likelihoods for two independent data points can be computed like this: Suppose we have two independent and identically distributed data points 5 and 10. Then, assuming that

the Normal distribution they come from has mean 10 and standard deviation 2, the joint likelihood of these is:

```
dnorm(5,mean=10,sd=2)*dnorm(10,mean=10,sd=2)
```

```
## [1] 0.001748
```

It is easier to do this on the log scale, because then one can add instead of multiplying. This is because $\log(x \times y) = \log(x) + \log(y)$. For example:

```
log(2*3)
```

```
## [1] 1.792
```

```
log(2) + log(3)
```

```
## [1] 1.792
```

So the joint log likelihood of the two data points is:

```
dnorm(5,mean=10,sd=2,log=TRUE)+dnorm(10,mean=10,sd=2,log=TRUE)
```

```
## [1] -6.349
```

Even more compactly:

```
sum(dnorm(c(5,10),mean=10,sd=2,log=TRUE))
```

```
## [1] -6.349
```

- Given the 10 data points above, calculate the maximum likelihood estimate (MLE) of the expectation.
- The sum of the log-likelihoods of the data x, using as the mean the MLE from the sample, and standard deviation 5.
- What is the sum of the log-likelihood if the mean used to compute the log-likelihood is `495.6`?

- Which value for the mean, the MLE or `495.6`, gives the higher log-likelihood?

1.9.5 Generating bivariate data

Generate 50 data points from two random variables X and Y, where $X \sim Normal(50, 100)$ and $Y \sim Normal(100, 20)$. The correlation between the random variables is 0.7. Plot the simulated data points from Y against those from X.

1.9.6 Generating multivariate data

The bivariate case can be generalized to more than two dimensions. Generate 50 data points from three random variables X, Y, and Z, where $X \sim Normal(50, 100)$, $Y \sim Normal(100, 20)$, and $Z \sim Normal(200, 50)$. The correlation between the random variables X and Y is 0.5, between X and Z is 0.2, an between Y and Z is 0.7. Here, you will have to define a 3×3 variance covariance matrix, with the pairwise covariances in the off-diagonals. Plot the simulated data points as two-dimensional figures: Y against X, Y against Z, and X against Z.

2

Hypothetical repeated sampling and the t-test

This chapter introduces some of the foundational ideas behind hypothesis testing in the frequentist framework. The key idea is that of hypothetical repeated sampling. When we fit a model to a given data-set, we are assuming that this is one of potentially infinite numbers of exact repetitions of an experiment. We will leverage some amazing properties of these exact repetitions in order to draw inferences from our particular data. The key idea to understand here is the central limit theorem, which we will discuss below. Before we do that, it is useful to introduce some terminology relating to typical experiment designs in linguistics and psychology.

2.1 Some terminology surrounding typical experiment designs in linguistics and psychology

An experiment typically involves taking a *random sample* from some population. For example, we could take a sample of participants and record their reading times for two types of sentences, for example easy and difficult sentences. How one operationalizes easy and difficult is not important at this point. However, to make the discussion concrete, consider this well-known example from psycholinguistics. We could be comparing reading times in active vs. passive sentences:

Active sentence: The man threw the ball.

Passive sentence: The ball was thrown by the man.

One claim in the literature is that, due to their simpler structure, active sentences are easier to process than passive sentences. So

this is an example of a simpler vs. complex sentence. If we want to investigate whether actives are easier to process than passives, we can, as an example, compare the total reading times for each sentence type (there are some problems that arise here because the sentences don't have the same length, so the comparison is not really fair; these issues we will discuss later).

Technically, we are supposed to randomly choose participants; in practice, this randomization rarely happens. Instead we take whatever participants we get, such as university students who happen to apply to participate in an experiment! So, random sampling is an assumption in all the discussions in this chapter, but in practice this assumption may or may not be perfectly met.

One design decision that the researcher must take is whether to collect only one response from each participant in each condition, or whether we collect multiple measures from each participant. If we collect only one data point from each participant, we will say that the data points are *independent* of each other. When we collect multiple measurements from each participant, we will say that we have *dependent* data; such multiple measurements are also called *repeated measures*. With repeated measures, there will be some correlation between the data-points because the multiple measurements from a common source.

In psycholinguistics, repeated measures experiments are the norm. Furthermore, it is common to use a so-called *Latin Square* design. To understand this design, suppose we want to conduct an experiment with two conditions, a and b, and that we want each participant to see each condition multiple times (repeated measurements). The way such an experiment (with two conditions) is set up with a Latin Square design is that we would create multiple pairs of items, e.g., items 1a, 1b, 2a, 2b, etc. For example, we are investigating active vs. passive sentences, item 1a could be the active clause *The man threw the ball*, and 1b would thene the passive counterpart *The ball was thrown by the man*. The two versions of the item are therefore minimal pairs—minimally different in that the sentence has the same words, but one is an active clause and the other a passive clause.

Then, we would create two groups of items, G1 and G2. Group G1 would contain items 1a, 2b, 3a, 4b, etc; group G2 would contain items 1b, 2a, 3b, 4a, etc. Thus, if one had 16 items, each group would contain eight instance of each condition. Having set up these two groups, one would then randomly assign each participant to one of the two groups. In this way, from each participant, we would obtain eight repeated measures from each condition. One can (and should) of course randomize the order of presentation in each group, possibly randomizing afresh for each participant. The term Latin Square refers to the fact that the ordering of the conditions forms a square.

a	b
b	a

Each condition appears in each row exactly once.

The Latin square generalizes beyond two conditions easily. Suppose we have four conditions; then, the Latin Square is:

a	b	c	d
b	c	d	a
c	d	a	b
d	a	b	d

Similarly if you have eight conditions, the Latin Square table would look like this.

a	b	c	d	e	f	g	h
b	c	d	e	f	g	h	a
c	d	e	f	g	h	a	b
d	e	f	g	h	a	b	c
e	f	g	h	a	b	c	d
f	g	h	a	b	c	d	e
g	h	a	b	c	d	e	f
h	a	b	c	d	e	f	g

As an aside, it is rarely a good idea to design such complex experiments! The more conditions you have in an experiment, the more decisions you will have to make about how to analyze the data, and the harder the interpretation of the data. Later in this course, we will demonstrate the price one has to pay for setting up overly complex designs. We mention this point here because it is a common beginner error to want to have as many conditions as possible in a single experiment. In fact, a student of ours once proposed a 180 condition design!

The Latin Square design is attractive for psycholinguistics and psychology because it has some optimality properties. Experiment design and the properties of different designs is a whole field in itself in statistics, but we won't get into these details.

One consequence of the above repeated measurements design is that we have multiple measurements not just from participants but also from items! Thus, items can also be seen as producing dependent data. In other words, both participants and items are producing dependent data in this design. Later on, we will see that this has an effect on the type of data analysis one can do.

With this very brief introduction to experiment design, we now turn to the idea of hypothetical repeated sampling, and the central limit theorem.

2.2 The central limit theorem using simulation

Suppose we collect some data, which can be represented by a vector y; this is a *single sample*. Given data y, and assuming for concreteness that the underlying likelihood is a $Normal(\mu = 500, \sigma = 100)$, the sample mean and standard deviation, $\bar{y}$ and s give us an estimate of the unknown parameters mean μ and the standard deviation σ of the distribution from which we assume that our data come from. Figure 2.1 shows the distribution of a particular sample, where the number of data points is $n = 1000$. Note that in this example the parameters are specified by us, so they are not

unknown; in a real data-collection situation, the sample mean and standard deviation are all we have as estimates of the parameters.

```
## sample size:
n<-1000
##  independent and identically distributed sample:
y<-rnorm(n,mean=500,sd=100)
## histogram of data:
hist(y,freq=FALSE)
## true value of mean:
abline(v=500,lwd=2)
```

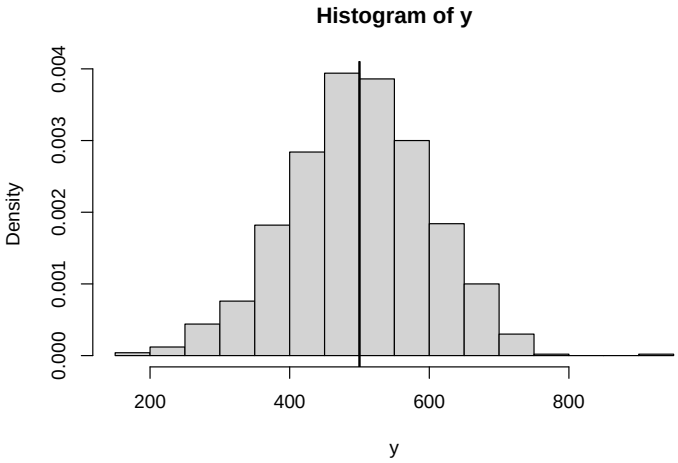


FIGURE 2.1: A sample of data y of size n=100, from the distribution Normal(500,100). The vertical line shows the true mean of the distribution we are sampling from.

Suppose now that you had not a single sample of size 1000 but many repeated samples. This isn't something one can normally do in real life; we often run a single experiment or, at most, repeat the same experiment once. However, one can simulate repeated sampling easily within R. Let us take 100 repeated samples like the one above, and save the samples in a matrix containing n=1000 rows and 100 columns, each column representing an experiment:

```
mu<-500
sigma<-100
## number of experiments:
k<-100
## store for data:
y_matrix<-matrix(rep(NA,n*k),ncol=k)
for(i in 1:k){
  ## expt result with sample size n:
  y_matrix[,i]<-rnorm(n,mean=mu,sd=sigma)
}
```

Now, if we compute the means $\bar{y}_k$ of *each* of the $k = 1, \ldots, 100$ experiments we just carried out, if certain conditions are met, these means will be normally distributed, with mean μ and standard deviation $\sigma/\sqrt{n}$. To understand this point, it is useful to first visualize the distribution of means and graphically summarize this standard deviation, which confusingly is called *standard error*.

```
## compute means from each replication:
y_means<-colMeans(y_matrix)
## the mean and sd (=standard error) of the means
mean(y_means);sd(y_means)
```

```
## [1] 499.8
```

```
## [1] 3.067
```

The sampling distribution of means has a normal distribution provided two conditions are met: (a) the sample size should be "large enough", and (b) μ and σ are defined for the probability density or mass function that generated the data. This fact is called the **central limit theorem** (CLT). The significance of the CLT for us as researchers is that from the summary statistics computed from a *single* sample, we can obtain an estimate of this distribution of means: $Normal(\bar{y}, s/\sqrt{n})$.

The statement that the sampling distribution of the means will

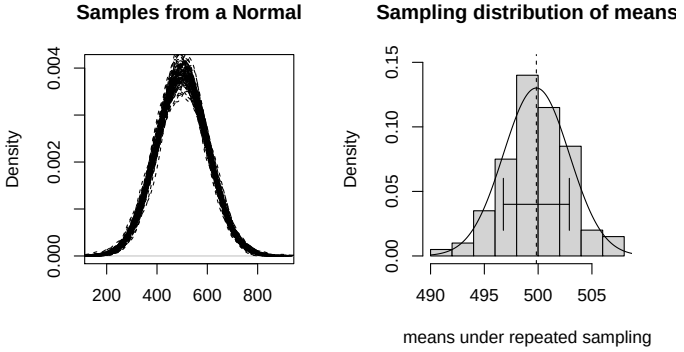


FIGURE 2.2: Sampling from a normal distribution (left); and the sampling distribution of the means under repeated sampling (right). The right-hand plot shows an overlaid normal distribution, and the standard deviation (standard error) as error bars.

be normal, with mean μ and standard deviation $\sigma/\sqrt{n}$, can be derived formally through a surprisingly simple application of random variable theory. Suppose we gather independent and identically distributed data $y_1, \ldots, y_n$, each of which is generated by a random variable $Y \sim Normal(\mu, \sigma)$.

When we compute the mean $\bar{y}$ for each sample, we are assuming that each of the means is coming from a random variable $\bar{Y}$, which is just a linear combination of values generated by instances of the random variable Y, which itself has a pdf with mean (expectation) μ and variance σ^2:

$$\bar{Y} = \frac{1}{n}\sum_{i=1}^{n} Y = \frac{1}{n}Y_1 + \cdots + \frac{1}{n}Y_n \tag{2.1}$$

So, the expectation of $\bar{Y}$ is

$$\begin{aligned} E[\bar{Y}] =& E[\frac{1}{n}Y_1 + \cdots + \frac{1}{n}Y_n] \\ =& \frac{1}{n}(E[Y] + \cdots + E[Y]) \\ =& \frac{1}{n}(\mu + \cdots + \mu) \\ =& \frac{1}{n}n\mu \\ =& \mu \end{aligned} \tag{2.2}$$

And the variance of $\bar{Y}$ is

$$\begin{aligned} Var(\bar{Y}) =& Var(\frac{1}{n}Y_1 + \cdots + \frac{1}{n}Y_n) \\ =& \frac{1}{n^2}Var(Y_1 + \cdots + Y_n) \end{aligned} \tag{2.3}$$

The last line above arises because the variance of a random variable Z multiplied by a constant a, $Var(aZ)$ is $a^2Var(Z)$. Here, $a = 1/n$, so $a^2 = 1/n^2$. Because $Y_1, \ldots, Y_n$ are independent, we can compute the variance $Var(Y_1 + \cdots + Y_n)$ by using the fact that the variance of the sum of independent random variables is the sum of their variances. This fact gives us:

$$\begin{aligned} \frac{1}{n^2}Var(Y_1 + \cdots + Y_n) =& \frac{1}{n^2}(Var(Y) + \cdots + Var(Y)) \\ =& \frac{1}{n^2}nVar(Y) \\ =& \frac{1}{n}Var(Y) \\ =& \frac{\sigma^2}{n} \end{aligned} \tag{2.4}$$

This derives the above result that the expectation (i.e., the mean) and variance of the sampling distribution of the sample means are

$$E[\bar{Y}] = \mu \quad Var(\bar{Y}) = \frac{\sigma^2}{n} \tag{2.5}$$

The above means that we can estimate the expectation $\bar{Y}$ and the variance of $\bar{Y}$ from a *single* sample!!

The Central Limit Theorem, not proved here (for a proof, see p. 267 of Miller and Miller (2004)) can be summarized as follows.

Central Limit Theorem

Let $f(Y)$ be the pdf of a random variable Y, and assume that the pdf has mean μ and variance σ^2. Then:

$$\bar{Y} \sim Normal(\mu, \sigma^2/n) \quad E[Y] = \mu, Var(Y) = \sigma^2 \quad \text{when n is large} \tag{2.6}$$

For us, the practical implication of this result is huge. From a *single* sample of n independent data points $y_1, \ldots, y_n$, we can derive the distribution of *hypothetical* sample means under repeated sampling. That is, it becomes possible to say something about what the plausible and implausible values of the sample mean are under repeated sampling. This is the basis for all hypothesis testing and statistical inference in the frequentist framework that we will look at in this book.

Sometimes the central limit theorem is misunderstood to imply that the pdf $f(Y)$ that is assumed to generate the data is always going to be normal. It is important to understand that there are two pdfs we are talking about here. First, there is the pdf that the data were generated from; this need not be normal. For example, you could get data from a Normal, Exponential, Gamma, or other distribution. Second, there is the sampling distribution of the *sample mean* under repeated sampling. It is the sampling distribution that the central limit theorem is about, not the distribution that generated the data.

2.3 Three examples of the sampling distribution

In the above discussion, the underlying pdf we sampled from above was a normal distribution. However, it need not be. Consider two examples first: the underlying pdf is an Exponential or a Gamma distribution.

The Exponential distribution has a parameter λ (parameterized in `R` as a `rate`, $1/\lambda$); its mean is λ and its variance is $1/\lambda^2$. The sampling distribution is normal, even though the underlying distribution is an Exponential; see Figure 2.3.

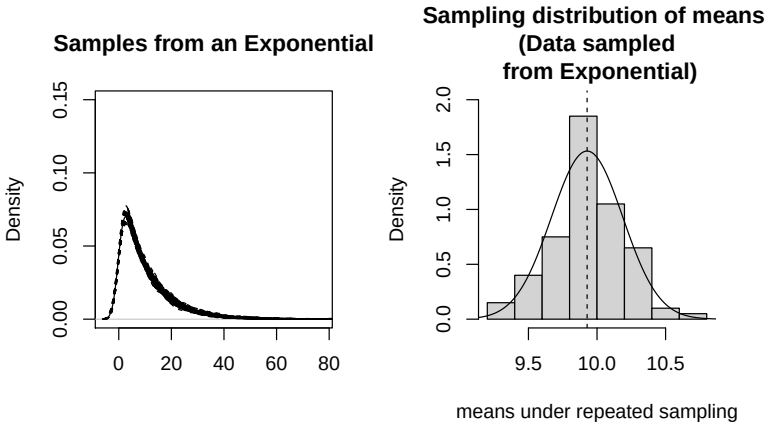


FIGURE 2.3: Sampling from an exponential.

A further example is samples from a Gamma distribution. Suppose we sample from a Gamma distribution with shape parameter chosen arbitrarily to be 1. The distribution of means is again going to be approximately normal; see Figure 2.4.

As a final example, consider what happens if sample from a distribution, the Cauchy, that doesn't have any mean or variance defined for it.

As the Figure @(fig:sdsmcauchy) illustrates, when the mean and variance for the likelihood are undefined, the central limit theorem doesn't hold. In the rest of this book, we will always assume that

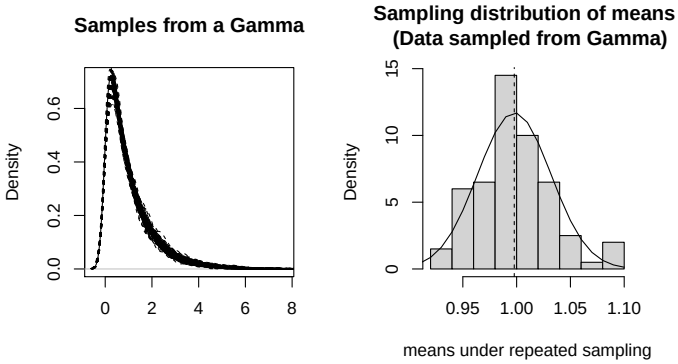


FIGURE 2.4: Sampling from a Gamma distribution.

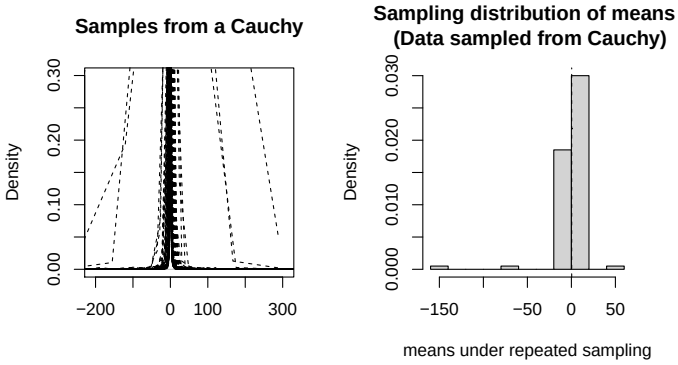


FIGURE 2.5: Sampling from a Cauchy distribution.

the data are coming from a distribution that has a mean and variance defined for it.

2.4 The confidence interval, and what it's good for

Once we have sample of data y, and once the sample mean $\bar{y}$ and the $SE = s/\sqrt{n}$ have been computed, it is common to define a so-called 95% confidence interval:

$$\bar{y} \pm 2SE \tag{2.7}$$

Because the sampling distribution of means is normally distributed, and because 95% of the area under the curve is covered by two times the standard deviation of the normal distribution, the upper and lower bounds of the interval defined by the interval $\bar{y} \pm 2SE$ covers approximately 95% of the area under the curve in the sampling distribution.

This interval is usually computed after estimating the sample mean and standard error from a data-set, and is called the *confidence interval* (CI). It has the following meaning: If you take samples repeatedly and compute the CI each time, 95% of those CIs will contain the true population mean μ. To understand this point, one can simulate this situation. This time we will do 1000 repeated experiments instead of 100.

```
mu<-500
sigma<-100
n<-1000
nsim<-1000
lower<-rep(NA,nsim)
upper<-rep(NA,nsim)
for(i in 1:nsim){
  y<-rnorm(n,mean=mu,sd=sigma)
  lower[i]<-mean(y) - 2 * sd(y)/sqrt(n)
  upper[i]<-mean(y) + 2 * sd(y)/sqrt(n)
}

## check how many CIs contain mu:
CIs<-ifelse(lower<mu & upper>mu,1,0)
## approx. 95% of the CIs contain true mean:
round(mean(CIs),2)
```

```
## [1] 0.96
```

Figure 2.6 visualizes the coverage properties of the confidence interval in 100 simulations; by coverage we mean here the proportion of cases where the true μ is contained in the CI.

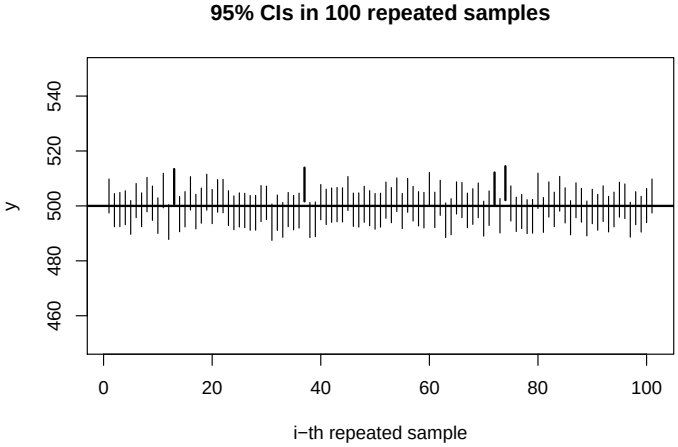


FIGURE 2.6: Illustration of the meaning of a 95 percent confidence interval (CI). The thicker bars represent the CIs which do not contain the true mean.

The confidence interval is widely misinterpreted, i.e., as representing the range of plausible value of the μ parameter. This is the wrong interpretation because μ is a point value by assumption, it doesn't have a pdf associated with it. The frequentist CI is defined with reference to the sampling distribution of the mean under repeated sampling, not the probability distribution of μ. By contrast, the Bayesian credible interval does have this interpretation. In most modeling settings we have encountered in our work, the frequentist confidence interval and Bayesian credible interval have very similar widths, with the Bayesian interval being slightly wider depending on the prior specifications.

Given the convoluted meaning of the CI, and the impossibility of interpreting a single CI, it is reasonable to ask: what good is a CI? One can treat the CI as a summary that tells us the width of the sampling distribution of the mean—the wider the sampling distribution, the more the implied variability under repeated sampling. The confidence interval can therefore be used to assess how uncertain we can be about the estimate of the sample mean under hypothetical repeated sampling. See Cumming (2014) for a useful perspective relating to using confidence intervals for inference. As

discussed later in the book, we will use the CI to informally assess uncertainty.

We turn next to the central ideas behind the hypothesis test. We begin with the humble one-sample t-test, which contains many subtleties and is well worth close study before we move on to the main topic of this book: linear mixed models.

2.5 Hypothesis testing: The one sample t-test

With the central limit theorem and the idea of hypothetical repeated sampling behind us, we turn now to one of the simplest statistical tests that one can do with continuous data: the t-test.

Due to its simplicity, it is tempting to take only a cursory look at the t-test and move on immediately to the linear (mixed) model. This would be a mistake. The humble t-test is surprising in many ways, and holds several important lessons for us. There are subtleties in this test, and a close connection to the linear mixed model. For these reasons, it is worth slowing down and spending some time understanding this test. Once the t-test is clear, more complex statistical tests will be easier to follow, because the logic of these more complex tests will essentially be more of the same, or variations on this general theme. You will see later that t-test can be seen as an analysis of variance or ANOVA; and the paired t-test is exactly the linear mixed model with varying intercepts.

2.5.1 The one-sample t-test

As in our running example, suppose we have a random sample y of size n, and the data come from a $N(\mu, \sigma)$ distribution, with unknown parameters μ and σ. An assumption of the t-test is that the data points are independent in the sense discussed at the beginning of this chapter. We can estimate μ from the sample mean $\bar{y}$, which we will sometimes also write as $\hat{\mu}$. We can also estimate σ from the sample standard deviation s, which we can also write

as $\hat{\sigma}$. These estimates in turn allow us to estimate the sampling distribution of the mean under (hypothetical) repeated sampling:

$$N(\hat{\mu}, \frac{\hat{\sigma}}{\sqrt{n}}) \tag{2.8}$$

It is important to realize here that the above sampling distribution is only as realistic as the estimates of the mean and standard deviation parameters—if those happen to be inaccurately estimated, then the sampling distribution is not realistic either.

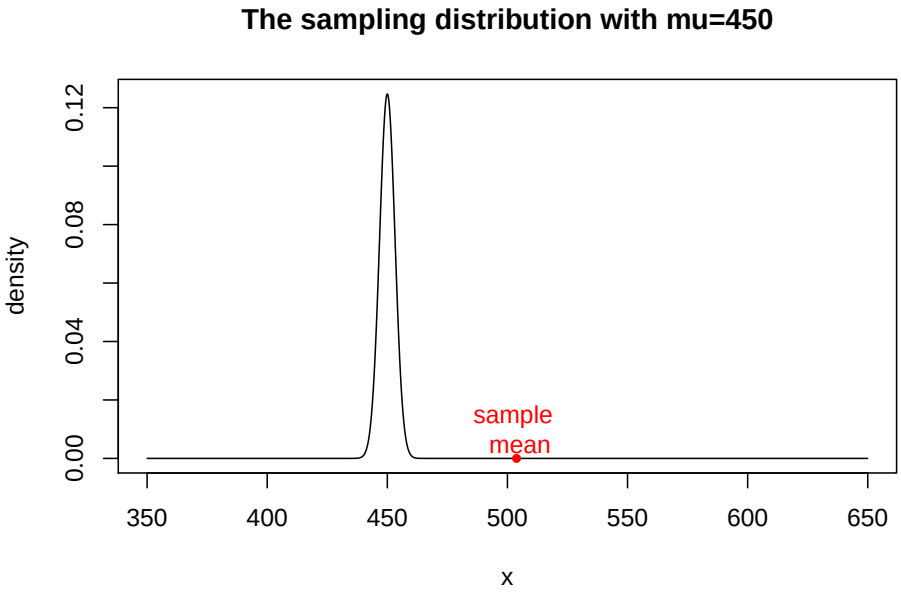
Assume as before that we take an independent random sample of size 1000 from a random variable Y that is normally distributed, with mean 500 and standard deviation 100. As usual, begin by estimating the mean and SE:

```
n<-1000
mu<-500
sigma<-100
## generate simulated data:
y <- rnorm(n,mean=500,sd=100)
## compute summary statistics:
y_bar<-mean(y)
SE<-sd(y)/sqrt(n)
```

The null hypothesis significance testing (NHST) approach as practised in psychology and other areas is to set up a null hypothesis that μ has some fixed value. Just as an example, assume that our null hypothesis is:

$$H_0 : \mu = 450 \tag{2.9}$$

This amounts to assuming that the true sampling distribution of sample means is (approximately) normally distributed and centered around 450, with the standard error estimated from the data.



The intuitive idea here is that

- if the sample mean $\bar{y}$ is "near" the hypothesized μ (here, 450), the data are possibly (but by no means necessarily) consistent with the null hypothesis distribution.
- if the sample mean $\bar{y}$ is "far" from the hypothesized μ, the data are inconsistent with the null hypothesis distribution.

The terms "near" and "far" will be quantified by determining how many standard error units the sample mean is from the hypothesized mean. This way of thinking shifts the focus away from the sampling distribution above, towards the distance measured in standard error units from the null hypothesis mean.

The distance between the sample mean and the hypothesized mean can be written in SE units as follows. We will say that the sample mean is t standard errors away from the hypothesized mean:

$$t \times SE = \bar{x} - \mu \tag{2.10}$$

If we divide both sides with the standard error, we obtain the so-called observed t-statistic:

$$t = \frac{\bar{x} - \mu}{SE} \tag{2.11}$$

This observed t-value, an expression of the distance between the sample mean and the hypothesized mean, becomes the basis for the statistical test.

Notice that the t-value is a random variable: it is a transformation of $\bar{X}$, the random variable generating the sample means. The t-value can therefore be seen as an instance of the following transformed random variable T:

$$T = \frac{\bar{X} - \mu}{SE} \tag{2.12}$$

This random variable has a pdf associated with it, the t-distribution, which is defined in terms of the sample size n; the pdf is written $t(n-1)$. Under repeated sampling, the t-distribution is generated from this random variable T.

We will compactly express the statement that "the observed t-value is assumed to be generated (under repeated sampling) from a t-distribution with n-1 degrees of freedom" as:

$$T \sim t(n-1) \tag{2.13}$$

For large n, the pdf of the random variable T approaches $N(0,1)$. This is illustrated in Figure 2.7; notice that the t-distribution has fatter tails than the normal for small n, say $n < 20$, but for larger n, the t-distribution and the normal become increasingly similar in shape. Incidentally, when n=2, the t-distribution $t(1)$ is the Cauchy distribution we saw earlier; this distribution is characterized by fat tails, and has no mean or variance defined for it.

Thus, given a sample size n, we can define a t-distribution corresponding to the null hypothesis distribution. For large values of

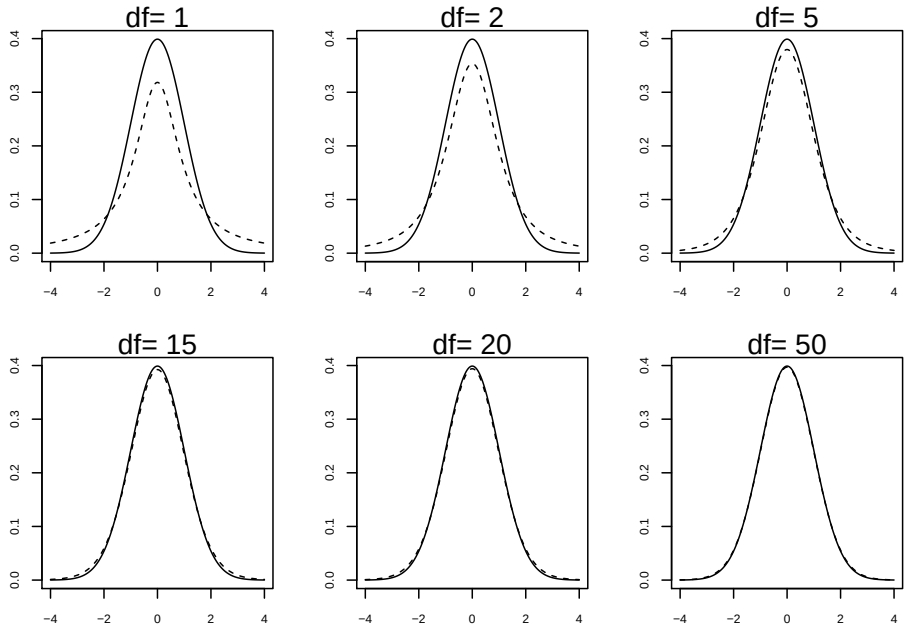


FIGURE 2.7: A visual comparison of the t-distribution (with degrees of freedom ranging from 1 to 50) with the standard normal distribution (N(0,1)). The dashed line represents the t-distribution, and the solid line the normal distribution.

n, we could even use $N(0, 1)$, although it is traditional in psychology and linguistics to always use the t-distribution no matter how large n is.

The null hypothesis testing procedure proceeds as follows:

- Define the null hypothesis: in our example, the null hypothesis was that $\mu = 450$. This amounts to making a commitment about what fixed value we think the true underlying distribution of sample means is centered at.
- Given data of size n, estimate $\bar{y}$, standard deviation s, and from that, estimate the standard error $s/\sqrt{n}$. The standard error will be used to describe the sampling distribution's standard deviation.
- Compute the observed t-value:

$$t = \frac{\bar{y} - \mu}{s/\sqrt{n}} \tag{2.14}$$

- Reject null hypothesis if the observed t-value is "large" (to be made more precise next).
- Fail to reject the null hypothesis, or (under some conditions, to be made clear later) even go so far as to accept the null hypothesis, if the observed t-value is "small".

What constitutes a large or small observed t-value? Intuitively, the t-value from the sample is large when we end up far in *either* tail of the distribution. The two tails of the t-distribution will be referred to as the *rejection region*. The word *region* here refers to the real number line along the x-axis, under the tails of the distribution. The rejection region will go off to infinity on the outer sides, and is demarcated by a vertical line on the inner side of each tail. This is shown in Figure 2.8. It goes off to infinity because the support—the range of possible values—of the random variable that the t-distribution belongs to stretches from minus infinity to plus infinity.

The location of the vertical lines is determined by the so-called *critical t-value* along the x-axis of the t-distribution. This is the value such that the area under the curve in the tails to the left or right of the tails is 0.025. As discussed in chapter 1, this area under the curve represents the probability of observing a value as extreme as the critical t-value, or some value that is more extreme. Notice that if we ask ourselves what the probability is of observing some particular t-value (a point value), the answer must necessarily be 0 (if you are unclear about why, re-read chapter 1). But we can ask the question: what is the absolute t-value, written $|t|$, such that $P(T > |t|) = 0.025$? That's the critical t-value. We call it the critical t-value because it demarcates the rejection region shown in Figure 2.8: we are adopting the convention that any observed t-value larger than this critical t-value allows us to reject the null hypothesis.

For a given sample size n, we can identify the rejection region

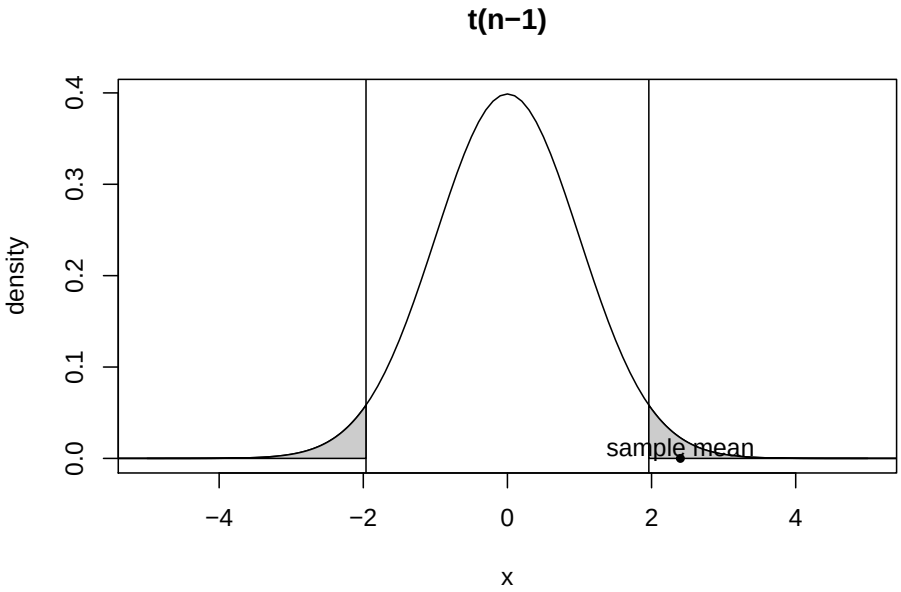


FIGURE 2.8: The rejection region in a t-distribution. The rejection region is the x-axis under the gray-colored area.

by using the `qt` function, whose usage is analogous to the `qnorm` function, discussed in chapter 1.

Because the shape of the t-distribution depends on the degrees of freedom (n-1), the critical t-value beyond which we reject the null hypothesis will change depending on sample size. For large sample sizes, say $n > 50$, the rejection point is about 2.

```
abs(qt(0.025,df=15))
```

```
## [1] 2.131
```

```
abs(qt(0.025,df=50))
```

```
## [1] 2.009
```

Consider the observed t-value from our sample in our running example:

```
## null hypothesis mean:
mu<-450
(t_value<-(y_bar-mu)/SE)
```

```
## [1] 16.82
```

This observed t-value is huge and is telling you the distance of the sample mean from the null hypothesis mean μ in standard error units.

$$t = \frac{\bar{y} - \mu_0}{s/\sqrt{n}} \text{ or } t\frac{s}{\sqrt{n}} = \bar{y} - \mu_0 \tag{2.15}$$

For large sample sizes, if the absolute t-value $|t|$ is greater than 2, we will reject the null hypothesis.

For a smaller sample size n, you can compute the exact critical t-value:

```
qt(0.025,df=n-1)
```

```
## [1] -1.962
```

Why is this critical t-value negative in sign? That is because it is on the left-hand side of the t-distribution, which is symmetric. The corresponding value on the right-hand side is:

```
qt(0.975,df=n-1)
```

```
## [1] 1.962
```

These values are of course identical if we ignore the sign. This is why we always frame our discussion around the absolute t-value.

In R, the built-in function `t.test` delivers the observed t-value. Given our running example, with the null hypothesis $\mu = 450$, R returns the following:

```
## observed t-value from t-test function:
t.test(y,mu=450)$statistic
```

```
##     t
## 16.82
```

The default value for the null hypothesis mean μ in this function is 0; so if one doesn't define a null hypothesis mean, the statistical test is done with reference to a null hypothesis that $\mu = 0$. That is why this t-value does not match our calculation above:

```
t.test(y)$statistic
```

```
##     t
## 157.5
```

In the most common usage of the t-test, the null hypothesis mean will be 0, because usually one is comparing a difference in means between two conditions or two sets of conditions. So the above line of code will work out correctly in those cases; but if you ever have a different null hypothesis mean than 0, then you have to specify it in the `t.test` function.

So, the way that the t-test is used in psychology and relate areas is to implement a *decision*: either reject the null hypothesis or fail to reject it. Whenever we do an experiment and carry out a t-test, we use the t-test to make this binary decision: reject or fail to reject the null hypothesis.

Recall that behind the t-test lies the assumption that the observed t-value is coming from a random variable, $T \sim t(n-1)$. The particular t-value we observe from a particular data-set belongs to a distribution of t-values under hypothetical repeated sampling. Thus, implicit in the logic of the t-test—and indeed every frequentist statistical test—is the assumption that the experiment is in principle repeatable: the experiment can in principle be re-run as many times as we want, assuming we have the necessary resources and time.

A quick simulation of t-values under repeated sampling makes this clear. Suppose that our null hypothesis mean is 450, and our sample size $n = 1000$. Assume that the data come from $Normal(\mu = 450, \sigma = 100)$. Let's do 10000 simulations, compute the sample mean each time, and then store the observed t-value. The t-distribution that results is shown in Figure 2.9.

```
n<-1000
nsim<-10000
tvals<-rep(NA,nsim)
for(i in 1:nsim){
y<-rnorm(n,mean=450,sd=100)
SE<-sd(y)/sqrt(n)
tvals[i]<-(mean(y)-450)/SE
}
plot(density(tvals),main="Simulated t-distribution",xlab="t-values under rep
```

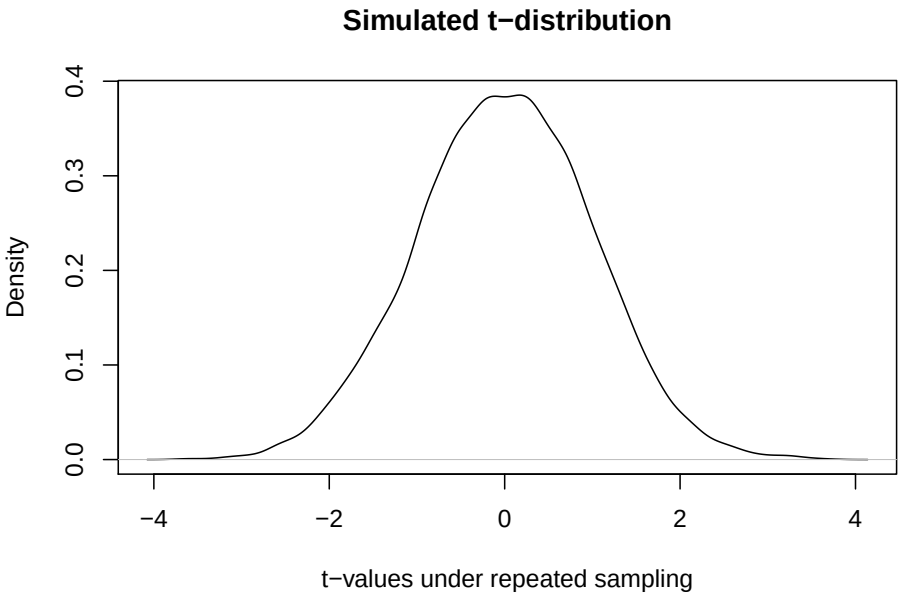


FIGURE 2.9: The distribution of t-values under repeated sampling.

This implicit idea of the experiment's repeatability leads to an

	Possible world 1	**Possible world 2**
	H_0 TRUE: $\mu = \mu_0$	H_0 FALSE $\mu = \mu_{alt}$
Decision: 'reject':	α Type I error	$1 - \beta$ Power
Decision: 'fail to reject':	$1 - \alpha$	β Type II error

TABLE 2.4: The possible realities (null is true or null is false) and the possible decisions (accept or reject null) we can take based on our observed t-value.

important aspect of the t-test: the proportion of times we would reject the null hypothesis under repeated sampling, given that the null is in fact true or false. In other words, we will now study the t-test's ability (at least in theory) to lead the researchers to the correct decision under (hypothetical) repeated sampling. We turn to this issue next.

2.5.2 Type I, II error, and power

When we do a hypothesis test using the t-test, the observed t-value will either fall in the rejection region, leading us to reject the null hypothesis, or it will land in the non-rejection region, leading us to fail to reject the null. That is a single, one-time event.

So suppose we have made our decision based on the observed t-value. Now, the null hypothesis can be either true or not true; we don't know which of those two possibilities is the reality. When we decide (based on the observed t-value) that the null is true, we are asserting that the parameter μ actually does have the hypothesized value μ_0; when we decide that the null is false, we are asserting that the parameter μ has some *specific* value μ_{alt} other than μ_0. We can represent these two alternative possible realities in a tabular form, as shown in Table 2.4. The two columns show the two possible worlds, one in which the null is true, and the other in which it is false. The two rows show the two possible decisions we can take based on the observed t-value: reject the null or fail to reject it.

As the table shows, we can make two kinds of mistakes:

- Type I error or α: Reject the null when it's true.
- Type II error or β: Accept the null when it's false.

In psychology and related areas, Type I error is usually fixed a priori at 0.05. This stipulated Type I error value is why the absolute critical t-value is kept at approximately 2; if, following recommendations from Benjamin et al. (2018), we were to stipulate that the Type I error be 0.005, then the critical t-value would have had to be set at:

```
abs(qt(0.0025,df=n-1))
```

```
## [1] 2.813
```

This suggested change in convention hasn't been taken up yet in cognitive science, but this could well change one day.

Type II error, the probability of incorrectly accepting the null hypothesis when it is false with some particular value for the parameter μ, is conventionally recommended (Cohen, 1988) to be kept at 0.20 or lower. This implies that the probability of correctly rejecting a null hypothesis for some particular true value of μ is 1-Type II error. This probability, called statistical power, or just power, should then obviously be larger than 0.80. Again, there is nothing special about these stipulations; they are conventions that became the norm over time.

Next, we will consider the trade-off between Type I and II error. For simplicity, assume that the standard error is 1, and the null hypothesis is that $\mu = 0$. This means that the observed t-value is really the sample mean.

Consider the concrete situation where, in reality, the true value of μ is 2. As mentioned above, the null hypothesis H_0 is that $\mu = 0$. Now the H_0 is false because $\mu = 2$ and not 0. Type I and II error can be visualized graphically as shown in Figure 2.10.

To understand Figure 2.10, one has to consider two distributions

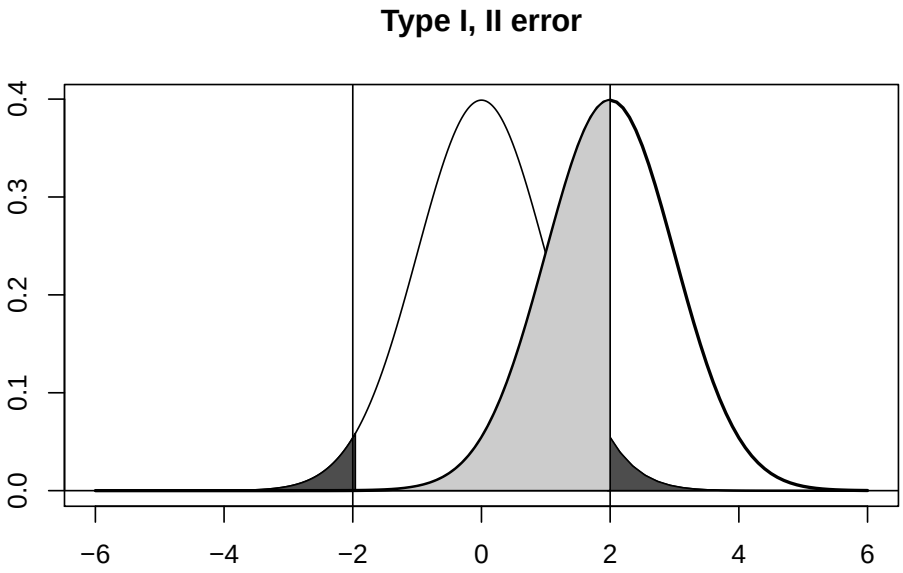


FIGURE 2.10: A visualization of Type I and II error. The dark-shaded tails of the left-hand side distribution represent Type I error; and the light-colored shaded region of the right-hand side distribution represents Type II error. Power is the unshaded area under the curve in the right-hand side distribution.

side by side. First, consider the null hypothesis distribution, centered at 0. Under the null hypothesis distribution, the rejection region lies below the dark colored tails of the distributions. The area under the curve in these dark-colored tails is the Type I error (conventionally set at 0.05) that we decide on before we even conduct the experiment and collect the data. Because the Type I error is set at 0.05, and because the t-distribution is symmetric, the area under the curve in each tail is 0.025. The absolute critical t-value helps us demarcate the inner boundaries of the rejection regions through the vertical lines shown in the figure. These vertical lines play a crucial role in helping us understand Type II error and power. This becomes clear when we consider the distribution representing the alternative possible value of μ, the distribution centered around 2. In this second distribution, consider now the area under the curve between the vertical lines demarcating the rejection region under the null. This area under the curve is the

probability of accepting the null hypothesis when the null hypothesis is false with some specific value (here, when μ has value 2).

Some interesting observations follow. Suppose that the true effect is in fact $\mu = 2$, as in the above illustration. Then,

- Simply decreasing Type I error to a smaller value like 0.005 will increase Type II error, which means that power (1-Type II error) will fall.
- Increasing sample size will squeeze the vertical lines closer to each other because standard error will go down with increasing sample size. This will reduce Type II error, and therefore increase power. Decreasing sample size will have the opposite effect.
- If we design an experiment with a larger effect size, e.g., by setting up a stronger manipulation (concete examples will be discussed in this book later on), our Type II error will go down, and therefore power will go up. Figure 2.11 shows a graphical visualization of a situation where the true effect size is $\mu = 4$. Here, Type II error is much smaller compared to Figure 2.10, where $\mu = 2$.

In summary, when we plan out an experiment, we are also required to specify the Type I and II error associated with the design. Both sources of error are within our control, at least to some extent. The Type I error we decide to use will determine our critical t-value and therefore our decision criterion for rejecting, failing to reject, or even (under certain conditions, to be discussed below) accepting the null hypothesis.

The Type II error we decide on will determine the long-run probability of incorrectly accepting the null hypothesis; its inverse (1-Type II error), statistical power, will determine the long-run probability of correctly rejecting the null hypothesis under the assumption that the μ has some particular assumed value.

That's the theory anyway. In practice, researchers only rarely consider the power properties of their experiment design; the focus is almost exclusively on Type I error. The neglect of power in experiment design has had interesting consequences for theory de-

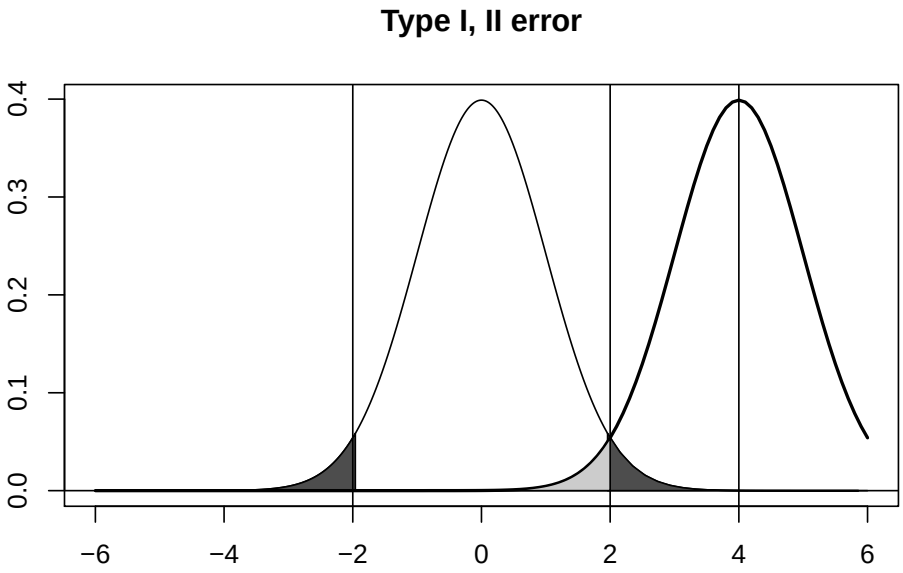


FIGURE 2.11: The change in Type II error if the true effect has mean 4.

velopment, as we will see later in this book. For a case study in psycholinguistics, see Vasishth et al. (2018).

2.5.3 How to compute power for the one-sample t-test

Power (which is 1-Type II error) is a function of three variables:

- the effect size
- the standard deviation
- the sample size.

There are two ways that one can compute power in connection with the t-test: either one can use the built-in R function, `power.t.test`, or one can use simulation.

2.5.3.1 Power calculation using the power.t.test

Suppose that we have an expectation that an effect size is 15 ms ± 5 ms (this could be based on the predictions of a theoretical model, or prior data); suppose also that prior experiments show standard deviations ranging from 100 to 300 ms. This is enough

information to compute a power curve as a function of effect size and standard deviation. See Figure 2.12 and the associated code below.

```
sds<-seq(100,300,by=1)
lower<-power.t.test(delta=15-5,sd=sds,n=10,strict =TRUE)$power
upper<-power.t.test(delta=15+5,sd=sds,n=10,strict =TRUE)$power
meanval<-power.t.test(delta=15,sd=sds,n=10,strict =TRUE)$power

plot(sds,meanval,type="l",
     main="Power curve (n=10)\n
     using power.t.test",
     xlab="standard deviation",
     ylab="power")
lines(sds,lower,lty=2)
lines(sds,upper,lty=2)
text(125,0.053,"10")
text(150,0.054,"15")
text(175,0.056,"20")
```

2.5.3.2 Power calculations using simulation

An analogous calculation as the one shown above using the `power.t.test` function can also be done using simulated data. First, generate simulated data repeatedly for each possible combination of parameter values (here, effect size and standard deviation), and compute the proportion of significant effects for each parameter combination. This can be done by defining a function that takes as input the number of simulations, sample size, effect size, and standard deviation:

```
compute_power<-function(nsim=1000,n=10,
                        effect = NULL,
                        stddev=NULL
                        ){
  temp_power<-rep(NA,nsim)
```

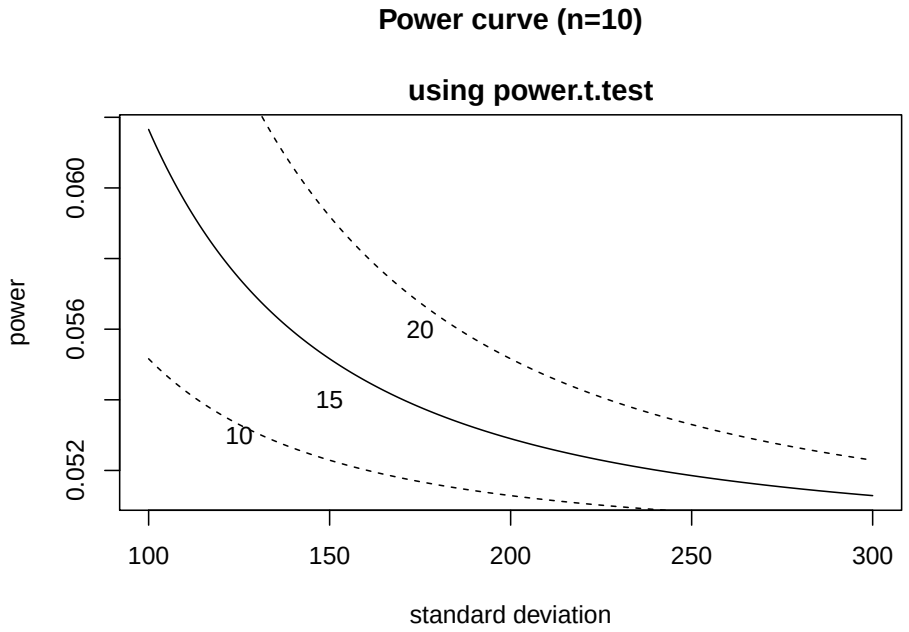


FIGURE 2.12: An illustration of a power curve for 10 participants, as a function of standard deviation, and three estimates of the effect: 15, 10, and 20.

```
  for(i in 1:nsim){
   y<-rnorm(n,mean=effect,sd=stddev)
   temp_power[i]<-ifelse(abs(t.test(y)$statistic)>2,1,0)
  }
  ## return power calculation:
  mean(temp_power)
}
```

Then, plot the power curves as a function of effect size and standard deviation, exactly as in Figure 2.12. Power calculations using simulations are shown in Figure 2.13. It is clear that simulation-based power estimation is going to be noisy; this is because each time we are generating simulated data and then carrying out a statistical test on it. This is no longer a closed-form mathematical calculation as done in `power.t.test` (this function simply

implements a formula for power calculation specified for this simple case). Because the power estimates will be noisy, we show a smoothed lowess line for each effect size estimate.

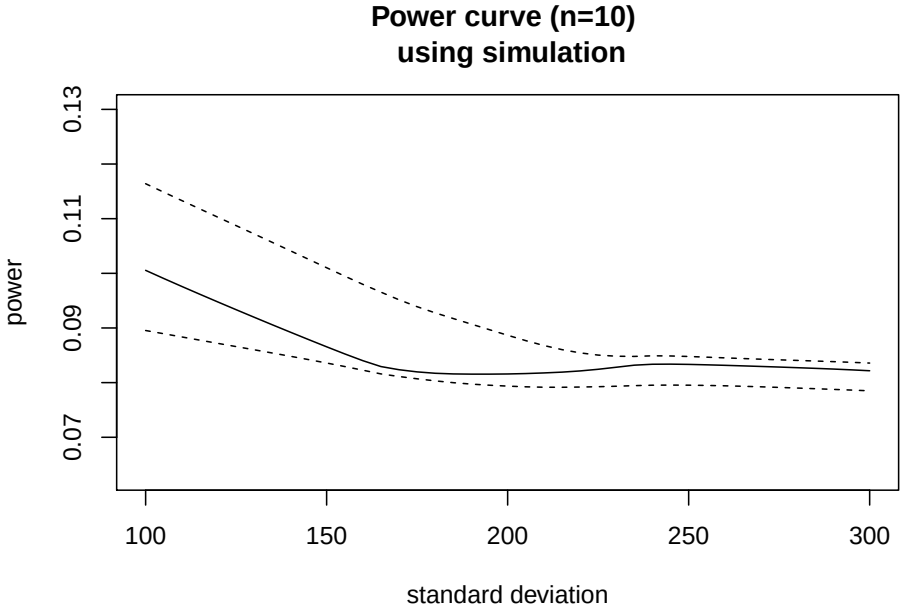


FIGURE 2.13: An illustration of a power curve using simulation, for 10 participants, as a function of standard deviation, and three estimates of the effect: 15, 10, and 20. The power curves are lowess-smoothed.

In the above example, simulation-based power calculation is overkill, and completely unnecessary because we have `power.t.test`. However, the technique shown above will be extended and will become our bread-and-butter method once we switch to power calculations for complicated linear mixed models. There, no closed form calculation can be done to compute power, at least not without oversimplifying the model; simulation will be the only practical way to calculate power.

It is important to appreciate the fact that power is a *function*; it isn't a single number. Because we can never be sure what the true effect size is, or what the true standard deviation is, power

functions (power as a function of plausible values for the relevant parameters) are much more useful than single numbers.

2.5.4 The p-value

Continuing with our t-test example, the `t.test` function in R will not only print out a t-value as shown above, but also a probability known as a *p-value*. This is the probability of obtaining the observed t-value that we obtained, or some value more extreme than that, conditional on the assumption that the null hypothesis is true.

We can compute the p-value "by hand". This can be computed, as done earlier, simply by calculating the area under the curve that lies beyond the observed t-value. It is standard practice to take the tail probability on both sides of the t-distribution.

```
(t_value<-t.test(y,mu=450)$statistic)
```

```
##         t
## -0.008827
```

```
2*pt(abs(t_value),df=n-1,lower.tail=FALSE)
```

```
##      t
## 0.9931
```

The area from both sides of the tail is taken because it is conventional to do a so-called *two-sided t-test*: our null hypothesis is that $\mu = 450$, and our alternative hypothesis is two-sided: μ is either less than 450 or μ is larger than 450. When we reject the null hypothesis, we are accepting this alternative, that μ could be some value other than 450. Notice that this alternative hypothesis is remarkably vague; we would reject the null hypothesis regardless of whether the sample mean turns out to be 600 or -600, for example. The practical implication is that the p-value gives us the strength of the evidence against the null hypothesis; it doesn't give us evidence in favor of a specific alternative, such as saying that μ

is positive or negative in sign. In psychology and allied disciplines, whenever the p-value falls below 0.05, it is common practice to write something along the lines that "there was reliable evidence for the predicted effect." This statement is incorrect! We only ever have evidence against the null. By looking at the sample mean and its sign, we are making a very big leap that we have evidence for the specific sample mean we happened to get. As we will see below, the sample mean can be wildly far from the true mean that produced the data.

One need not have a two-sided alternative; one could have defined the alternative to be one-sided (for example, that $mu > 450$). In that case, one would compute only one side of the area under the curve. This kind of one-sided test is not normally done, but one can imagine a situation where a one-sided test is justified (for example, when only one sign of the effect is possible, or if there is a strong theoretical reason to expect only one particular sign—positive or negative—on an effect). That said, in their scientific career, only one of the authors of this book has ever had occasion to use a one-sided test. In this book, we will not use one-sided t-tests.

The p-value is always interpreted with reference to the pre-defined Type I error. Conventionally, we reject the null if $p < 0.05$. This is because we set the Type I error at 0.05. Keep in mind that Type I error and the p-value are two distinct things. Type I error is the probability of your incorrectly rejecting the null under repeated sampling. This is not the same thing as your p-value. The latter will be obtained from a particular experiment, and will vary from experiment to experiment; it is a random variable. Type I error is a value we fix in advance.

2.5.4.1 *The distribution of the p-value under the null hypothesis

We have been talking about a continuous random variable as a dependent measure, and have learnt about the standard two-sided t-test, with a point null hypothesis. When we do such a test, we usually use the p-value to decide whether to reject the null hypothesis or not.

Sometimes, you will hear statisticians (e.g., Andrew Gelman on

his blog) criticize p-values by saying that the null hypothesis significance test is a "specific random number generator". What does that sentence mean? We explain this point here because it is important for understanding the meaning of the p-value.

Suppose that the null hypothesis is in fact true. We will now simulate the distribution of the p-value under repeated sampling:

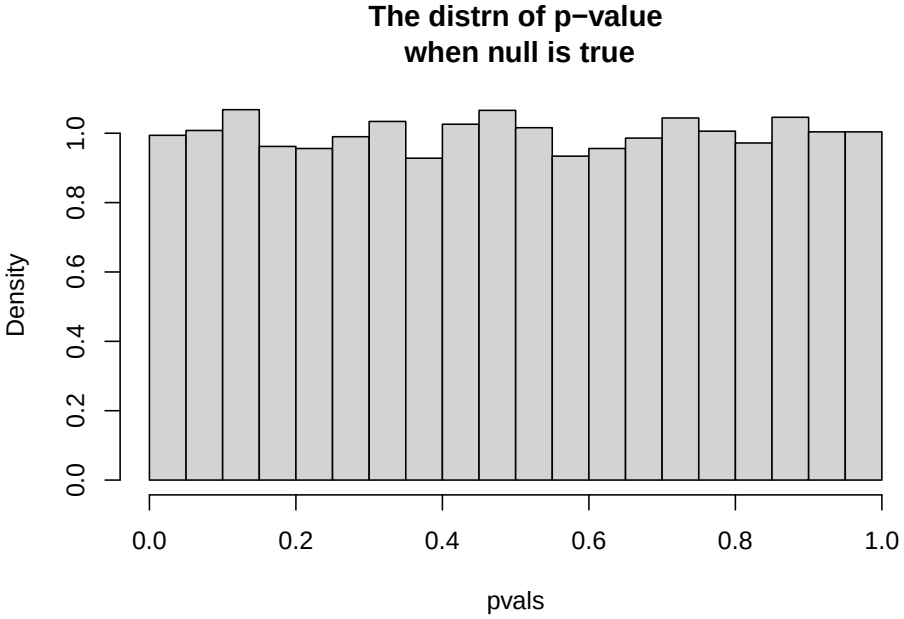


FIGURE 2.14: The p-value has a uniform distribution when the null hypothesis is true; a demonstration using simulation.

The distribution of the p-value is uniform—every value between 0 and 1 is equally likely.

We will now formally derive the above fact, that the distribution of the p-value is uniform under the null hypothesis.

Consider the fact that the p-value is a random variable; call it Z. The p-value is the cumulative distribution function (CDF) of the random variable T, which itself is a transformation of the random variable $\bar{Y}$:

$T = (\bar{X} - \mu)/(\sigma/\sqrt{n})$

This random variable T has some CDF $F(T)$. It is possible to show that if a random variable $Z = F(T)$, i.e., if Z is the CDF for the random variable T, then Z has a uniform distribution ranging from 0 to 1, $Z \sim Uniform(0, 1)$.

This is an amazing fact. To get a grip on this, let's first think about the fact that when a random variable Z comes from a $Uniform(0, 1)$ distribution, then $P(Z < z) = z$. Consider some examples:

- when $z = 0$, then $P(Z < 0) = 0$;
- when $z = 0.25$, then $P(Z < 0.25) = 0.25$;
- when $z = 0.5$, then $P(Z < 0.5) = 0.5$;
- when $z = 0.75$, then $P(Z < 0.75) = 0.75$;
- when $z = 1$, then $P(Z < 1) = 1$.

Next, we will prove the above statement, that if a random variable $Z = F(T)$, i.e., if Z is the CDF for a random variable T, then $Z \sim Uniform(0, 1)$. The proof is actually quite astonishing and even has a name: it's called the *probability integral transform*.

Suppose that Z is the CDF of a random variable T: $Z = F(T)$. Then, it follows that $P(Z \leq z)$ can be rewritten in terms of the CDF of T: $P(F(T) \leq z)$. Now, if we apply the inverse of the CDF (F^{-1}) to both the left and right sides of the inequality, we get $P(F^{-1}F(T) \leq F^{-1}(z))$. But $F^{-1}F(T)$ gives us back T; this holds because if we have a one-to-one onto function $f(x)$, then applying the inverse f^{-1} to this function gives us back x.

The fact that $F^{-1}F(T)$ gives us back T means that we can rewrite $P(F^{-1}F(T) \leq F^{-1}(z))$ as $P(T \leq F^{-1}(z))$. But this probability is simply the CDF $F(F^{-1}(z))$, which simplifies to z. This shows that $P(Z \leq z) = z$; i.e., that the p-value has a uniform distribution under the null hypothesis.

The above proof is restated below compactly:

$$\begin{aligned} P(Z \leq z) =& P(F(T) \leq z) \\ =& P(F^{-1}F(T) \leq F^{-1}(z)) \\ =& P(T \leq F^{-1}(z)) \\ =& F(F^{-1}(z)) \\ =& z \end{aligned} \tag{2.16}$$

It is for this reason that statisticians like Andrew Gelman periodically point out that "the null hypothesis significance test is a specific random number generator". The practical implication of this criticism of p-values is that when we do a single experiment and obtain a p-value under the assumption that the null is true, if the null were in fact true, then we are just using a random number generator to make a decision that the effect is present or absent. A broader implication is that we should not place our theory development exclusively at the feet of the p-value. As we discuss in this book, other considerations (such as replicability, uncertainty of the estimates, and power) are as or even more important.

2.5.5 Type M and S error in the face of low power

Beyond Type I and II error, there are also two other kinds of error to be aware of. These are Type M and S error; both sources of error are closely related to statistical power.

The terms Type M and S error were introduced by Gelman et al. (2014), but the ideas have been in existence for some time (Hedges, 1984),(Lane and Dunlap, 1978). Button et al. (2013) refer to Type M and S error as the "winner's curse" and "the vibration of effects." In related work, Ioannidis (2008) refers to the vibration ratio in the context of epidemiology.

Type S and M error can be illustrated with the following example. Suppose your true effect size is believed to be $D = 15$, then we can compute (apart from statistical power) the following error rates, which are defined as follows:

- **Type S error**: the probability that the sign of the effect is incorrect, given that the result is statistically significant.
- **Type M error**: the expectation of the ratio of the absolute magnitude of the effect to the hypothesized true effect size, given that the result is significant. Gelman and Carlin also call this the exaggeration ratio, which is perhaps more descriptive than "Type M error".

Suppose that a particular study has standard error 46, and sample size 37. And suppose that the true $\mu = 15$, as in the example discussed above. Then, we can compute statistical power, Type S and M error through simulation in the following manner:

```
## probable effect size, derived from past studies:
D<-15
## SE from the study of interest:
se<-46
stddev<-se*sqrt(37)
nsim<-10000
drep<-rep(NA,nsim)
for(i in 1:nsim){
samp<-rnorm(37,mean=D,sd=stddev)
drep[i]<-mean(samp)
}
```

Power can be computed by simply determining the proportion of times that the absolute observed t-value is larger than 2:

```
##power: the proportion of cases where
## we reject the null hypothesis correctly:
(pow<-mean(ifelse(abs(drep/se)>2,1,0)))
```

```
## [1] 0.0589
```

Power is quite low here (we deliberately chose an example with low power to illustrate Type S and M error).

Next, we figure out which of the samples are statistically significant

(which simulated values yield $p < 0.05$). As a criterion, we use a t-value of 2 to declare that we reject the null; we could have done this more precisely by working out an exact critical t-value.

```
## which results in drep are significant at alpha=0.05?
signif<-which(abs(drep/se)>2)
```

Type S error is the proportion of significant cases with the wrong sign (sign error), and Type M error is the ratio by which the true effect (of $\mu = 15$) is exaggerated in those simulations that happened to come out significant.

```
## Type S error rate | signif:
(types_sig<-mean(drep[signif]<0))
```

```
## [1] 0.1902
```

```
## Type M error rate | signif:
(typem_sig<-mean(abs(drep[signif])/D))
```

```
## [1] 7.408
```

In this scenario, when power is approximately 6%, whenever we get a significant effect, the probability of obtaining the wrong sign is a whopping 19% and the effect is likely to be 7.4078 times larger than its true magnitude. The practical implication is as follows.

When power is low, relying on the p-value (statistical significance) to declare an effect as being present will be misleading because the decision will be based on an overestimate of the effect (Type M error), and even the sign of the effect could be wrong. This isn't just a theoretical point; it has real-world consequences for theory development. For an example from psycholinguistics regarding this point, see Vasishth et al. (2018).

Another useful way to visualize Type M and S error is through the so-called funnel plot. As shown in Figure 2.15, estimates obtained

from low-powered studies will tend to be exaggerated (the lower part of the funnel), and as power goes up, the effect estimates start to cluster tightly around the true value of the effect.

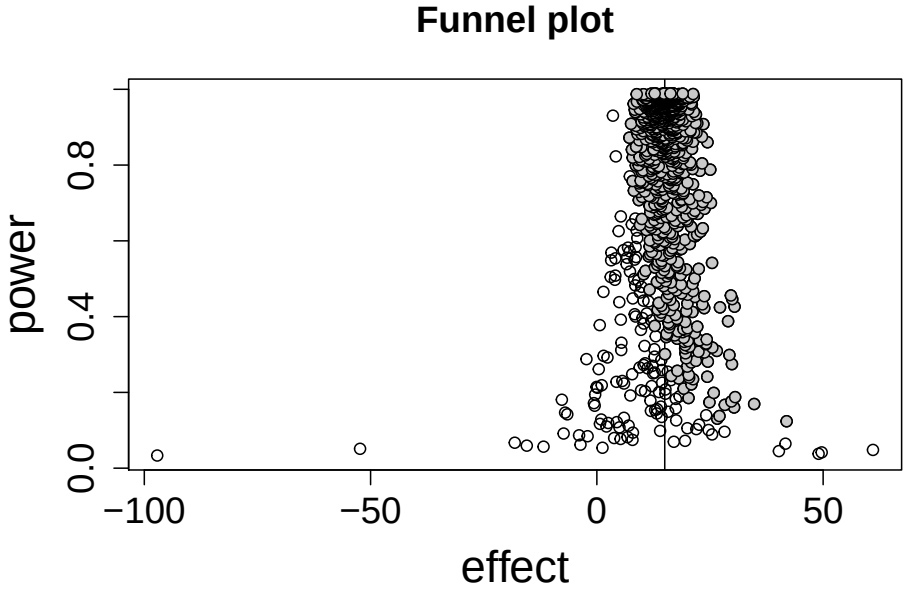


FIGURE 2.15: An illustration of a funnel plot. Shown are repeated samples of an effect estimate under different values of power, where the true value of the effect is 15 (marked by the vertical line). Significant effects are shaded gray. The lower the power, the wider the fluctuation of the effect; under low power, it is the exaggerated effects that end up statistically significant, even though they are very biased relative to the true value. As power goes up, the effect estimates start to cluster around the true value, and significant effects are also accurate estimates of the effect. Thus, low power leads to exaggerated estimates of the effect, especially if the data are filtered by statistical significance.

What is important to appreciate here is the fact that significant effects "point to the truth" just in case power is high; when power is low, either null results will frequently be found even if the null is false, and those results that turn out significant will be based on Type M error.

In many fields, it is practically impossible to conduct a high-

powered study. What should one do in this situation? When reporting results that are likely based on an underpowered study, the best approach is to openly acknowledge the power limitation, to attempt to conduct a direct replication of the effect to establish robustness, and to attempt to synthesize the evidence from existing knowlege (Cumming, 2014).

By direct replication, we mean that the study should be run multiple times with the same materials and design but new participants, to establish whether effect estimates in the original study and the replication study are consistent with each other. Direct replications stand in contrast to so-called conceptual replications, which are not exact repetitions of the original design, but involve some further or slightly different but related experimental manipulations. Conceptual replications are also a very useful tool for cross-validating the existence of an effect.

Direct replications will always differ from the original study in some way or another—the lab may differ, the protocols might differ slightly, the experimenter is different, etc. Such between-study variability is obviously unavoidable in direct-replication attempts, but they are still worthwhile for establishing the existence of an effect. To make more clear the idea of establishing robustness through replication attempts, detailed examples of different kinds of replication attempts of published studies will be presented in this book's example data-sets.

2.5.6 Searching for significance

The NHST procedure is essentially a decision procedure: if $p < 0.05$, we reject the null hypothesis; otherwise, we fail to reject the null. Because significant results are easier to publish than non-significant results, a common approach taken by researchers (including the first author of this book, when he was a graduate student) is to run the experiment and periodically check if statistical significance has been reached. The procedure can be described as follows:

- The experimenter gathers n data points, then checks for significance (is $p < 0.05$ or not?).
- If the result is not significant, he gets more data (say, n more data points). Then he checks for significance, and repeats.

Since time and money are limited, he might decide to stop collecting data after some multiple of n have been collected.

One can simulate different scenarios here. Suppose that n is initially 15.
Under the standard assumptions, we set Type I error to be 0.05. Let's suppose that the null hypothesis that $\mu = 0$ is in fact true, and that standard deviation is 250.

```
##Standard properties of the t-test:
pvals<-NULL
tstat_standard<-NULL
n<-15
nsim<-10000
## assume a standard dev of 1:
stddev<-250
mn<-0
for(i in 1:nsim){
  samp<-rnorm(n,mean=mn,sd=stddev)
  pvals[i]<-t.test(samp)$p.value
  tstat_standard[i]<-t.test(samp)$statistic
}
```

Type I error rate is about 5%, consistent with our expectations:

```
round(mean(pvals<0.05),2)
```

```
## [1] 0.05
```

But the situation quickly deteriorates as soon as we adopt the strategy outlined above. Below, we will also track the distribution of the t-statistic.

```
pvals<-NULL
tstat<-NULL
## how many subjects can I run?
upper_bound<-n*6

for(i in 1:nsim){
  significant<-FALSE
  x<-rnorm(n,mean=mn,sd=stddev) ## take sample
while(!significant & length(x)<upper_bound){
  ## if not significant:
if(t.test(x)$p.value>0.05){
  x<-append(x,rnorm(n,mean=mn,sd=stddev)) ## get more data
} else {significant<-TRUE}   ## otherwise stop:
}
pvals[i]<-t.test(x)$p.value
tstat[i]<-t.test(x)$statistic
}
```

Now, Type I error rate is much higher than 5%:

```
round(mean(pvals<0.05),2)
```

```
## [1] 0.15
```

Figure 2.16 shows the distributions of the t-statistic in the standard case vs with the above stopping rule:

What is important to realize here is that the inflation in Type I error we observed above was due to the fact that the t-distribution is no longer a t-distribution: we have bumps in the tails when we use the flexible stopping rule, and these raise our Type I error. This demonstrates why one should fix one's sample size in advance, based on a power analysis. One should not deploy a stopping rule like the one above; if we used such a stopping rule, we are much more likely to incorrectly declare a result as statistically significant than our intended Type I error rate of 0.05.

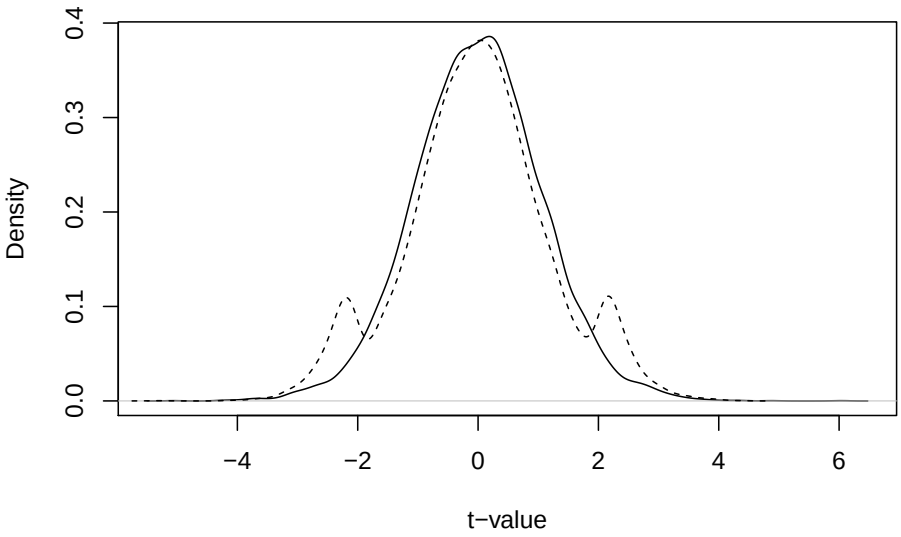


FIGURE 2.16: A comparison of the distribution of t-values with an a priori fixed stopping rule, versus a flexible stopping rule conditional on finding significance.

There can be compelling reasons to adopt the peek-and-run strategy; e.g., if one wants to avoid exposing patients to a treatment that might turn out to be harmful. In such situations, one can run an adaptive experimental trial by correcting for Type I error inflation (Pocock, 2013). In this book, we will aim to develop a workflow whereby the sample size is fixed through power analysis, in advance of running an experiment.

2.6 The two-sample t-test vs. the paired t-test

In our running example above, we examined the case where we have a single vector of data y. This led to the one-sample t-test.

Next, we consider a case where we have two vectors of data. The data-set below is from Johnson (2011). Shown below are F1 formant data (in Hertz) for different vowels produced by male and female speakers of different languages. (In a speech wave, different

bands of energy centered around particular frequencies are called formants.)

```
F1data<-read.table("data/F1_data.txt",header=TRUE)
F1data
```

```
##    female male vowel  language
## 1     391  339     i  W.Apache
## 2     561  512     e  W.Apache
## 3     826  670     a  W.Apache
## 4     453  427     o  W.Apache
## 5     358  291     i CAEnglish
## 6     454  406     e CAEnglish
## 7     991  706     a CAEnglish
## 8     561  439     o CAEnglish
## 9     398  324     u CAEnglish
## 10    334  307     i   Ndumbea
## 11    444  361     e   Ndumbea
## 12    796  678     a   Ndumbea
## 13    542  474     o   Ndumbea
## 14    333  311     u   Ndumbea
## 15    343  293     i      Sele
## 16    520  363     e      Sele
## 17    989  809     a      Sele
## 18    507  367     o      Sele
## 19    357  300     u      Sele
```

Notice that the male and female values can be seen as *dependent* or *paired*: each row belongs to the same vowel and language. Nevertheless, we can compare males' and females' F1 frequencies, completely ignoring this paired nature of the data. The t-test does not "know" whether these data are paired or not—it is the researcher's job to make sure that model assumptions are met. In this case, the assumption of the t-test is that the data are independent.

Let's ignore the paired nature of the data for now, and treat the two vectors as independent vectors. Suppose that our null hypothesis is that there is no difference between the mean F1's for males

(μ_m) and females (μ_f). Now, our null hypothesis is $H_0 : \mu_m = \mu_f$ or $H_0 : \mu_m - \mu_f = \delta = 0$.

This kind of design calls for a two-sample t-test.

The function call in R for a two-sample t-test is shown below. Note here that we are assuming that both the male and female F1 scores have equal variance.

```
t.test(F1data$female,F1data$male,
       paired=FALSE,
       var.equal=TRUE)
```

```
##
##  Two Sample t-test
##
## data:  F1data$female and F1data$male
## t = 1.5, df = 36, p-value = 0.1
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##  -30.07 217.54
## sample estimates:
## mean of x mean of y
##     534.6     440.9
```

This t-test is computing the following t-statistic:

$$t = \frac{d - (\mu_m - \mu_f)}{SE} = \frac{d - 0}{SE} \tag{2.17}$$

where d is the difference between the two sample means; the rest of the terms we are familiar with. SE is the standard error of the sampling distribution of the difference between the means.

We will now do this calculation "by hand". The only new things are the formula for the SE calculation, and the degrees of freedom for t-distribution $(2 \times n - 2) = 36$.

The standard error for the difference in the means in the two-sample t-test is computed using this formula:

$$SE_\delta = \sqrt{\frac{\hat{\sigma}_m^2}{n_m} + \frac{\hat{\sigma}_f^2}{n_f}} \tag{2.18}$$

Here, $\hat{\sigma}_m$ is estimate of the standard deviation for males, and $\hat{\sigma}_f$ for the females; the n are the respective sample sizes.

```
n_m<-n_f<-19
## difference of sample means:
d<-mean(F1data$female)-mean(F1data$male)
(SE<-sqrt(var(F1data$male)/n_m+var(F1data$female)/n_f))
```

```
## [1] 61.04
```

```
(observed_t <- (d-0)/SE)
```

```
## [1] 1.536
```

```
## p-value:
2*(1-pt(observed_t,df=36))
```

```
## [1] 0.1334
```

The output of the two-sample t-test and the hand-calculation above match up.

Now consider what will change once we take into account the fact that the data are paired. The two-sample t-test now becomes a so-called paired t-test.

For such paired data, the null hypothesis is as before: $H_0 : \delta = 0$. But since each row in the data-frame is paired (from the same vowel+language), we subtract the vector row-wise, and get a new *vector* d (not a single number d as in the two-sample t-test) with

the row-wise differences. Then, we just do the familiar one-sample test we saw earlier:

```
d<-F1data$female-F1data$male
t.test(d)
```

```
##
##  One Sample t-test
##
## data:  d
## t = 6.1, df = 18, p-value = 9e-06
## alternative hypothesis: true mean is not equal to 0
## 95 percent confidence interval:
##   61.48 125.99
## sample estimates:
## mean of x
##     93.74
```

An alternative syntax for the paired t-test explicitly feeds the two paired vectors into the function, but one must explicitly specify that they are paired, otherwise the test is a two-sample (i.e., unpaired) t-test:

```
t.test(F1data$female,F1data$male,paired=TRUE)
```

```
##
##  Paired t-test
##
## data:  F1data$female and F1data$male
## t = 6.1, df = 18, p-value = 9e-06
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##   61.48 125.99
## sample estimates:
## mean of the differences
##                   93.74
```

Incidentally, notice that the p-value in the paired t-test is statistically significant, unlike the two-sample t-test above. The null hypothesis is the same in both tests, but the significance level leads to different conclusions.

Which analysis is correct, the two-sample t-test or the paired t-test? It all depends on your assumptions about what the data represent. If you consider the data paired, for the reasons given above, then a paired test is called for. If there is no pairing (here, domain knowledge is required), we can treat this as unpaired data.

Next, we look at some perhaps subtle points about the paired t-test.

2.6.1 Common mistakes involving the t-test

The paired t-test assumes that each row in the data-frame is independent of the other rows. This implies that the data-frame cannot have more than one row for a particular pair. In other words, the data-frame cannot have repeated measurements spread out across rows.

For example, doing a paired t-test on this hypothetical data-frame would be incorrect:

female	male	vowel	language
391	339	i	W.Apache
400	320	i	W.Apache
⋮	⋮	⋮	⋮

Why? Because the assumption is that each row is independent of the others. This assumption is violated here (this is assuming that repeating the vowel from the same language will lead to some commonalities between the two repetitions).

Consider another hypothetical example. In the table below, from subject 1 we see two data points each for condition a and for condition b.

Here, we again have repeated measurements from subject 1. The independence assumption is violated.

condition a	condition b	subject	item
391	339	1	1
400	320	1	2
⋮	⋮	⋮	⋮

How to proceed when we have repeated measurements from each subject or each item? The solution is to aggregate the data so that each subject (or item) has only *one* value for each condition.

This aggregation allows us to meet the independence assumption of the t-test, but it has a potentially huge drawback: it pretends we have one measurement from each subject for each condition. Later on we will learn how to analyze unaggregated data, but if we want to do a paired t-test, we have no choice but to aggregate the data in this way.

A fully worked example will make this clear. We have repeated measures data on subject versus object relative clauses in English. Subject relative clauses are sentences like *The man who was standing near the doorway laughed*. Here, the phrase (called a relative clause) *who was standing near the doorway* modifies the noun phrase *man*; it is called a subject relative because the noun phrase *man* is the subject of the relative clause. By contrast, object relative clauses are sentences like *The man who was the woman was talking to near the doorway laughed*; here, the *man* is the grammatical object object of the relative clause *who was the woman was talking to near the doorway*.

The data are from a self-paced reading study reported in Grodner and Gibson (2005), their experiment 1. A theoretical prediction is that in English, object relatives are harder to read than subject relatives, in the relative clause verb region. We want to test this prediction.

First, load the data containing reading times from the region of interest (the relative clause verb):

```
gg05e1 <- read.table("data/grodnergibsonE1crit.txt",

                     header=TRUE)

head(gg05e1)
```

```
##    subject item condition rawRT
## 6        1    1    objgap   320
## 19       1    2   subjgap   424
## 34       1    3    objgap   309
## 49       1    4   subjgap   274
## 68       1    5    objgap   333
## 80       1    6   subjgap   266
```

We have repeated measurements for each condition from the subjects, and from items. You can establish this by using the `xtabs` command. Notice that there are no missing data points:

```
t(xtabs(~subject+condition,gg05e1))
```

```
##           subject
## condition 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18
##   objgap  8 8 8 8 8 8 8 8 8  8  8  8  8  8  8  8  8  8
##   subjgap 8 8 8 8 8 8 8 8 8  8  8  8  8  8  8  8  8  8
##           subject
## condition 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33
##   objgap   8  8  8  8  8  8  8  8  8  8  8  8  8  8  8
##   subjgap  8  8  8  8  8  8  8  8  8  8  8  8  8  8  8
##           subject
## condition 34 35 36 37 38 39 40 41 42
##   objgap   8  8  8  8  8  8  8  8  8
##   subjgap  8  8  8  8  8  8  8  8  8
```

```
t(xtabs(~item+condition,gg05e1))
```

```
##           item
```

```
## condition  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15
##   objgap  21 21 21 21 21 21 21 21 21 21 21 21 21 21 21
##   subjgap 21 21 21 21 21 21 21 21 21 21 21 21 21 21 21
##          item
## condition 16
##   objgap  21
##   subjgap 21
```

It is important to stress once more that it is the researcher's responsibility to make sure that the t-test's assumptions are met. For example, one could fit a two-sample t-test to the data as provided. The two-sample t-test can be implemented using the syntax shown below:

```
t.test(rawRT~condition,gg05e1)
```

```
##
##  Welch Two Sample t-test
##
## data:  rawRT by condition
## t = 3.8, df = 431, p-value = 2e-04
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##   48.98 155.59
## sample estimates:
##  mean in group objgap mean in group subjgap
##                 471.4                 369.1
```

This t-test is incorrect for several reasons, but the most egregious error here is that the data are paired (each subject delivers data for both conditions), and that property of the data is being ignored.

Another common mistake is to do a paired t-test on the data without checking that the data are independent in the sense discussed above. Again, the t.test function will happily return a meaningless result:

```
t.test(rawRT~condition,paired=TRUE,gg05e1)
```

```
##
##  Paired t-test
##
## data:  rawRT by condition
## t = 4, df = 335, p-value = 8e-05
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##   51.98 152.59
## sample estimates:
## mean of the differences
##                   102.3
```

Here, the degrees of freedom indicate that we have fit the incorrect model. There are 42 subjects and 16 items, and the presentation of items to subjects uses a Latin square design (each subject sees only one condition per item). The 335 degrees of freedom come from $42 \times 8 = 336$ data points, minus one. Why do we say 42×8 and not 42×16? That is because each subject will return eight differences in reading time for each condition: each subject gives us eight subject-relative data points and eight object-relative data points.

For each of the 42 subjects, the t-test function internally creates a vector of eight data points of subject relatives and subtracts the vector of eight data points of object relatives. That is how we end up with $42 \times 8 = 336$ data points.

These 336 data points are assumed by the t-test to be independent of each other; but this cannot be the case because each subject delivers eight data points for each condition; these are obviously dependent (correlated) because they come from the same subject.

What is needed is a *single* data-point for each subject and condition, and for each item and condition. In order to conduct the t-test, aggregation of the data by subjects and by items is necessary.

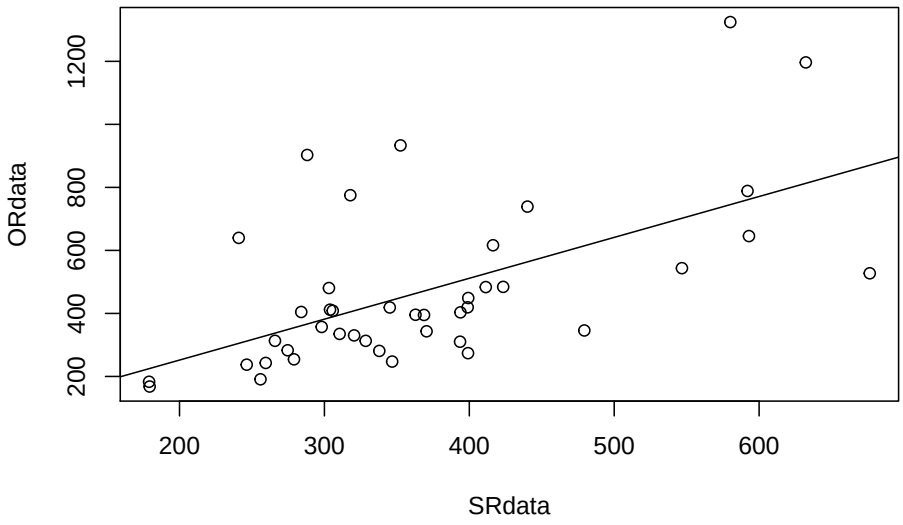
Consider the by-subjects aggregation procedure below. Now we have only one data-point for each condition and subject:

```
bysubj<-aggregate(rawRT~subject+condition,
                  mean,
                  data=gg05e1)
t(xtabs(~subject+condition,bysubj))
```

```
##           subject
## condition 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18
##   objgap  1 1 1 1 1 1 1 1 1  1  1  1  1  1  1  1  1  1
##   subjgap 1 1 1 1 1 1 1 1 1  1  1  1  1  1  1  1  1  1
##           subject
## condition 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33
##   objgap   1  1  1  1  1  1  1  1  1  1  1  1  1  1  1
##   subjgap  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1
##           subject
## condition 34 35 36 37 38 39 40 41 42
##   objgap   1  1  1  1  1  1  1  1  1
##   subjgap  1  1  1  1  1  1  1  1  1
```

Notice that the data are correlated: the longer the subject relative clause data from a participant, the longer their object relative clause data:

```
SRdata<-subset(bysubj,condition=="subjgap")$rawRT
ORdata<-subset(bysubj,condition=="objgap")$rawRT
plot(SRdata,ORdata)
abline(lm(ORdata~SRdata))
```




```
cor(SRdata,ORdata)
```

```
## [1] 0.5876
```

Returning to the t-test, by aggregating the data the independence assumption of the t-test is met, and the degrees of freedom for this by-subjects analysis are now correct ($42 - 1 = 41$):

```
t.test(rawRT~condition, bysubj,paired=TRUE)
```

```
##
##  Paired t-test
##
## data:  rawRT by condition
## t = 3.1, df = 41, p-value = 0.003
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##   35.85 168.72
## sample estimates:
## mean of the differences
##                   102.3
```

Similar to the by-subjects aggregation done above, one could do

a by-items aggregation and then a by-items t-test (What should be the degrees of freedom for the by-items analysis? There are 16 items in this data-set). This is left as an exercise for the reader.

The paired t-test illustrated above is actually not the best way to analyze this data-set, because it ignores the fact that each subject delivers not one but eight data points per condition. Each subject's repeated measurements will introduce a source of variance, but this source of variance is being suppressed in this t-test, leading to a possibly over-enthusiastic t-value. In order to take this variability into account, we must switch to the linear mixed model. But before we get to the linear mixed model, we have to consider the linear model. The next chapter turns to this topic.

2.7 Exercises

2.7.1 Practice using `qt`

Take an independent random sample of size `142` from a normal distribution with mean `123`, and standard deviation `70`. Next, we are going to pretend we don't know the population parameters (the mean and standard deviation). We compute the MLEs of the mean and standard deviation using the data and get the sample mean `145.242` and the sample standard deviation `50.885`.

- Compute the estimated standard error using the sample standard deviation provided above.
- What are your degrees of freedom for the relevant t-distribution?
- Calculate the **absolute** critical t-value for a 95% confidence interval using the relevant degrees of freedom you just wrote above.
- Next, compute the lower bound of the 95% confidence interval using the estimated standard error and the critical t-value.
- Finally, compute the upper bound of the 95% confidence interval using the estimated standard error and the critical t-value.

2.7.2 Computing the p-value

A paired t-test is done with data from 10 participants. The t-value from the test is 2.1. What is the p-value associated with a two-sided null hypothesis test?

2.7.3 Computing the t-value

If the p-value from a two-sided null hypothesis test had been 0.09, what would be the associated absolute t-value (i.e., ignoring the sign on the t-value)? The number of participants is 10, as above.

2.7.4 Type I and II error

Given that Type I error is 0.01; what is the highest value possible for Type II error?

2.7.5 Practice with the paired t-test

In a self-paced reading study, Grodner and Gibson (2005) investigated subjects vs. object relative clauses. They analyzed the reading times at the relative clause verb. However, a reviewer objects that the whole sentence's reading times (total reading times) should be used to evaluate the difference between the two conditions, because one cannot know where the difficulty might arise. It isn't clear whether one should use mean reading times over the entire sentence, or total reading times (summing up all the reading times over the entire sentence). Carry out a by-subjects paired t-test on (a) the critical relative clause verb, versus (b) mean reading time over all words in the two sentence types, and (c) total reading times over all words in the two sentence types. Compare the t-value across the three tests, and decide what the appropriate dependent variable might be (Note: there is no correct answer here).

The data are loaded and pre-processed as follows. The code below gives you the reading times for the critical verb. You will have to work out how to obtain mean or total reading times for the whole sentence in each condition.

```
## load data:
library(dplyr)
gg05e1 <- read.table("data/GrodnerGibson2005E1.csv",sep=",",
                     header=TRUE)
gge1 <- gg05e1 %>%
  filter(item != 0)

gge1 <- gge1 %>% mutate(word_positionnew = ifelse(item != 15 &
                                                    word_position > 10,
                                                  word_position-1, word_posi
#there is a mistake in the coding of word position,
#all items but 15 have regions 10 and higher coded
#as words 11 and higher

## get data from relative clause verb:
gge1crit <- subset(gge1, ( condition == "objgap" &
                             word_position == 6 ) |
            ( condition == "subjgap" & word_position == 4 ))
```

3

Linear models and linear mixed models

3.1 From the t-test to the linear (mixed) model

We begin with the Grodner and Gibson (2005) self-paced reading data we saw in the previous chapter. Load the data and compute the means for the raw reading times by condition:

```
gg05e1<-read.table("data/grodnergibsonE1crit.txt",
                   header=TRUE)
means<-round(with(gg05e1,tapply(rawRT,
                                IND=condition,
                                mean)))
means
```

```
##  objgap subjgap
##     471     369
```

As predicted by theory (Grodner and Gibson, 2005), object relatives (labeled objgap here) are read slower than subject relatives (labeled subjgap).

As discussed in the previous chapter, a paired t-test can be done to evaluate whether we have evidence against the null hypothesis that object relatives and subject relatives have identical reading times. However, we have to aggregate the data by subjects and by items first.

```
bysubj<-aggregate(rawRT~subject+
                    condition,
```

```
                    mean,data=gg05e1)
byitem<-aggregate(rawRT~item+
                      condition,
                    mean,data=gg05e1)
t.test(rawRT~condition,
       paired=TRUE,bysubj)$statistic
```

```
##     t
## 3.109
```

```
t.test(rawRT~condition,
       paired=TRUE,byitem)$statistic
```

```
##     t
## 3.754
```

What these two t-tests show is that both by subjects and by items, there is strong evidence against the null hypothesis that the object and relatives have identical reading times.

Interestingly, exactly the same t-values can be obtained by running the following commands, which implement a kind of linear model called the *linear mixed model*:

```
library(lme4)
```

```
## Loading required package: Matrix
```

```
##
## Attaching package: 'Matrix'
```

```
## The following objects are masked from 'package:tidyr':
##
##     expand, pack, unpack
```

```
##
## Attaching package: 'lme4'
```

```
## The following object is masked from 'package:SIN':
##
##     sdcor2cov
```

```
m0lmersubj<-lmer(rawRT~condition+(1|subject),bysubj)
summary(m0lmersubj)$coefficients
```

```
##                   Estimate Std. Error t value
## (Intercept)          471.4      31.13  15.143
## conditionsubjgap    -102.3      32.90  -3.109
```

```
m0lmeritem<-lmer(rawRT~condition+(1|item),byitem)
summary(m0lmeritem)$coefficients
```

```
##                   Estimate Std. Error t value
## (Intercept)          471.4      20.20  23.336
## conditionsubjgap    -102.3      27.25  -3.754
```

The signs of the t-values are the opposite to that of the paired t-tests above; the reason for that will presently become clear.

Our goal in this chapter is to understand the above model involving the `lmer` function, using the familiar paired t-test as a starting point.

For now, consider only the by-subject analysis. Given the sample means shown above for the two conditions, we can rewrite our best guess about how the object and subject relative clause reading time distributions were generated:

- Object relative: $Normal(471, \hat{\sigma})$
- Subject relative: $Normal(369, \hat{\sigma})$

This can also be rewritten with respect to the object relative mean and the difference between the two conditions (the reasons for this will become clear presently):

- Object relative: $Normal(471 - 102 \times 0, \hat{\sigma})$
- Subject relative: $Normal(471 - 102 \times 1, \hat{\sigma})$

Note that the two distributions for object and subject relative clauses (RCs) are assumed to be independent. This assumed independence is expressed by the fact that we define two separate Normal distributions, one for object relatives and the other for subject relatives. We saw earlier that this assumption of independence does not hold in our data, because we have one data point for each RC type from the same subject. However, for now we will ignore this detail; we will fix this shortcoming later.

The interesting point to notice here is that the mean for the object and subject relatives' distributions can be rewritten as a sum of two terms. A completely equivalent way to express the fact that object relatives are coming from a $Normal(471, \hat{\sigma})$ is to say that each object relative data-point can be described by the following equation:

$$y = 471 + -102 \times 0 + \varepsilon \text{ where } \varepsilon \sim Normal(0, \hat{\sigma}) \tag{3.1}$$

Similarly, the subject relative's distribution can be written as being generated from:

$$y = 471 - 102 \times 1 + \varepsilon \text{ where } \varepsilon \sim Normal(0, \hat{\sigma}) \tag{3.2}$$

In these data, the parameter $\hat{\sigma}$ is estimated to be 213. How do we know what this estimate is? This parameter's estimate can be derived from the by-subjects t-test output above: The observed t-value is

$$obs.t = \frac{\bar{x}}{s/\sqrt{n}} \tag{3.3}$$

Solving for s:

$$s = \bar{x} \times \sqrt{n}/obs.t = -102 \times \sqrt{42}/-3.109 = 213 \tag{3.4}$$

So, our model for the relative clause data consists of two equations:

Object relatives:

$$y = 471 - 102 \times 0 + \varepsilon \text{ where } \varepsilon \sim Normal(0, 213) \tag{3.5}$$

Subject relatives:

$$y = 471 - 102 \times 1 + \varepsilon \text{ where } \varepsilon \sim Normal(0, 213) \tag{3.6}$$

The above statements describe a *generative process* for the data.

Given such a statement about the generative process, we can express the estimated mean reading times for each RC type as follows. We can ignore the term ε because it has mean 0 (we stipulate this when we specify that $\varepsilon \sim Normal(0, \sigma)$).

Mean object relative reading times:

$$\text{Mean OR RT} = 471 - 102 \times 0 \tag{3.7}$$

Mean subject relative reading times:

$$\text{Mean SR RT} = 471 - 102 \times 1 \tag{3.8}$$

There is a function in R, the `lm()` function, which expresses the above statistical model, and prints out exactly the same numerical values that we used above:

```
summary(m0<-lm(rawRT~condition,bysubj))$coefficients
```

```
##                  Estimate Std. Error t value  Pr(>|t|)
## (Intercept)         471.4      31.13  15.143 1.795e-25
## conditionsubjgap   -102.3      44.02  -2.324 2.263e-02
```

The linear model function `lm()` prints out two coefficients, 471 and −102, that help express the mean reading times for object and subject relative data, using a simple coding scheme: object relatives

are coded as 0, and subject relatives are coded as 1. This coding scheme is not visible to the user, but is represented internally in R. The user can see the coding for each condition level by typing:

```
## make sure that the condition column is of type factor:
bysubj$condition<-factor(bysubj$condition)
contrasts(bysubj$condition)
```

```
##         subjgap
## objgap        0
## subjgap       1
```

We will discuss contrast coding in detail in a later chapter, but right now the simple 0,1 coding above—called treatment contrasts—is enough for our purposes.

Thus, what the linear model above gives us is two numbers: the mean object relative reading time (471), and the *difference* between object and subject relative (-102). We can extract the two coefficients by typing:

```
round(coef(m0))
```

```
##      (Intercept) conditionsubjgap
##              471             -102
```

In the vocabulary of linear modeling, the first number is called the *intercept*, and the second one is called the *slope*. Note that the meaning of the intercept and slope depends on the ordering of the factor levels. We can make the sample mean of the subject relative represent the intercept:

```
## reverse the factor level ordering:
bysubj$condition<-factor(bysubj$condition,
                         levels=c("subjgap","objgap"))
contrasts(bysubj$condition)
```

```
##          objgap
## subjgap       0
## objgap        1
```

Now, the intercept is the mean of the subject relatives, and the slope is the difference between object and subject relatives reading times. Note that the sign of the t-value has changed—the sign depends on the contrast coding.

```
m1a<-lm(rawRT~condition,bysubj)
summary(m1a)$coefficients
```

```
##                  Estimate Std. Error t value  Pr(>|t|)
## (Intercept)         369.1      31.13  11.857 1.819e-19
## conditionobjgap     102.3      44.02   2.324 2.263e-02
```

Let's switch back to the original factor level ordering:

```
bysubj$condition<-factor(bysubj$condition,
                         levels=c("objgap","subjgap"))
contrasts(bysubj$condition)
```

```
##          subjgap
## objgap         0
## subjgap        1
```

In mathematical form, the model can now be stated as a single equation:

$$rawRT = \beta_0 + \beta_1 condition + \varepsilon \tag{3.9}$$

where

- condition is a 0,1 coded vector, with object relatives coded as 0, and subject relatives coded as 1.
- β_0 is the mean for the object relative (which is coded as 0)
- β_1 is the amount by which the object relative mean must be changed to obtain the mean for the subject relative.

- ε is the noisy variation from trial to trial around the means for the two conditions, represented by $Normal(0, 213)$.

The null hypothesis of scientific interest here is always with reference to the slope, that the difference in means between the two relative clause types β_1 is:

$H_0 : \beta_1 = 0$

The t-test value printed out in the linear model is simply the familiar t-test formula in action:

$$obs.t = \frac{\beta_1 - 0}{SE} \tag{3.10}$$

The intercept also has a null hypothesis associated with it, namely that $H_0 : \beta_0 = 0$. However, this null hypothesis test is of absolutely no interest for us. This hypothesis test is reported by the lm() function only because the intercept is needed for technical reasons, to be discussed later.

The *contrast coding* mentioned above determines the meaning of the β parameters:

```
bysubj$condition<-factor(bysubj$condition,
                         levels=c("objgap","subjgap"))
contrasts(bysubj$condition)
```

```
##         subjgap
## objgap        0
## subjgap       1
```

When discussing linear models, we will make a distinction between the unknown true means β_0, β_1 and the estimated mean from the data $\hat{\beta}_0, \hat{\beta}_1$. The estimates that we have from the data are:

- Estimated mean object relative processing time: $\hat{\beta}_0 = 471$.
- Estimated mean subject relative processing time: $\hat{\beta}_0 + \hat{\beta}_1 = 471 + -102 = 369$.

3.2 Sum coding

We have established so far that the mathematical form of the model is:

$$rawRT = \beta_0 + \beta_1 condition + \varepsilon \tag{3.11}$$

We can change the contrast coding of the `condition` vector in the following way. First, recode the levels of the condition column as shown below.

```
## new contrast coding:
bysubj$cond<-ifelse(bysubj$condition=="objgap",1,-1)
```

Now, the two conditions are coded not as 0, 1 but as -1 and +1:

```
xtabs(~cond+condition,bysubj)
```

```
##     condition
## cond objgap subjgap
##   -1      0      42
##   1      42       0
```

With this coding, the model parameters have a different meaning:

```
m1<-lm(rawRT~cond,bysubj)
round(coef(m1))
```

```
## (Intercept)        cond
##         420          51
```

- The intercept now represents the grand mean processing time: $\hat{\beta}_0 = 420$.
- The mean object relative processing time is now: $\hat{\beta}_0 + \hat{\beta}_1 \times 1 = 420 + 51 = 471$.

- The mean subject relative processing time is: $\hat{\beta}_0 + \hat{\beta}_1 \times (-1) = 420 - 51 = 369$.

This kind of parameterization is called *sum-to-zero contrast* or more simply *sum contrast* coding. This is the coding we will use most frequently in this book. We will elaborate on contrast coding in a later chapter; there, the advantages of sum coding over treatment coding will become clear. For now, it is sufficient to understand that one can *reparametrize* the model using different contrast codings, and that such a reparametrization impacts the interpretation of the parameters.

With sum coding, the null hypothesis for the slope is

$$H_0 : \mathbf{1} \times \mu_{obj} + (-\mathbf{1}) \times \mu_{subj} = 0 \tag{3.12}$$

The sum contrast coding of +1 standing for object relatives and -1 standing for subject relatives in the linear model directly refer to the ± 1 coefficients in the null hypothesis above. Now the model is as follows.

Object relative reading times:

$$rt = 420 \times \mathbf{1} + 51 \times \mathbf{1} + \varepsilon \tag{3.13}$$

Subject relative reading times:

$$rt = 420 \times \mathbf{1} + 51 \times (-\mathbf{1}) + \varepsilon \tag{3.14}$$

One could write it in a single line as:

$$rt = 420 + 51 \times condition + \varepsilon \tag{3.15}$$

The term ε is call the residuals; it is the amount by which the observed data deviate from the values predicted by the above model. For example, suppose that a data point for a subject relative condition is 400 ms. The model predicts a reading time of 420-51=369

ms. The residual for that data point would be 400-369= 31. Another example: suppose that a data point for an object relative condition is 200 ms. The model predicts the object relative to be 420+51=471. The residual for that data point would then be 200-471=-271. Thus, the residual is the amount of the discrepancy between the model's predicted reading time and the actually observed reading time.

3.3 Checking model assumptions

It is an assumption of the linear model that the residuals are (approximately) normally distributed, That is what the statement $\varepsilon \sim Normal(0, \sigma)$ implies. It is important to check that model assumptions are approximately satisfied; this is because the null hypothesis significance testing procedure requires approximate normality of residuals.

Here is how we can check whether this normality assumption is met:

```
## extract residuals:
res.m1<-residuals(m1)
```

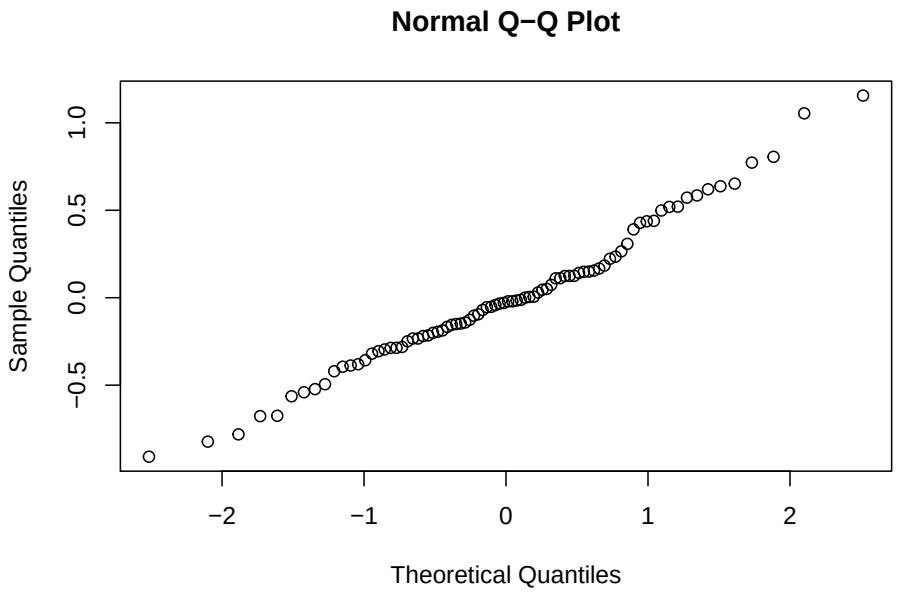
Compare the residuals to the quantiles of the standard normal distribution ($Normal(0, 1)$):

When the normality assumption is met, the residuals will align perfectly with the quantiles of the standard normal distribution, resulting in a straight diagonal line in the above plot. When the normality assumption is not met, the line will tend to curve away from the diagonal.

In the above case, a log transform of the data improves the normality of residuals. We will discuss transformation in detail later in this book; for now, it is sufficient to note that for continuous

data that consists of all-positive values (here, reading times), a log transform will often be the appropriate transform.

```
m1log<-lm(log(rawRT)~cond,bysubj)
qqnorm(residuals(m1log))
```



The estimates of the parameters are now in the log scale:

- The estimated grand mean processing time: $\hat{\beta}_0 = 5.9488$.
- The estimated mean object relative processing time: $\hat{\beta}_0 + \hat{\beta}_1 = 5.9488 + 0.0843 = 6.0331$.
- The estimated mean subject relative processing time: $\hat{\beta}_0 - \hat{\beta}_1 = 5.9488 - 0.0843 = 5.8645$.

The model does not change, only the scale does:

$$\log rt = \beta_0 + \beta_1 condition + \varepsilon \tag{3.16}$$

Now, the intercept and slope can be used to compute the reading time in the two conditions. Note that because $exp(log(rt)) = rt$, to get the mean estimates on the raw ms scale, we just need to exponentiate both sides of the equation:

$$exp(\log rt) = exp(\beta_0 + \beta_1 condition) \tag{3.17}$$

This approach gives us the following estimates on the ms scale:

- Estimated mean object relative reading time: $exp(\hat{\beta}_0 + \hat{\beta}_1) = exp(5.9488 + 0.0843) = 417$.
- Estimated mean subject relative reading time: $exp(\hat{\beta}_0 - \hat{\beta}_1) = exp(5.9488 - 0.0843 = 352$.

The difference in reading time is 417-352=65 ms. If we had fit the model to raw reading times, the difference would have been:

```
m1raw<-lm(rawRT~cond,bysubj)
```

- Estimated mean object relative reading time: $\hat{\beta}_0 + \hat{\beta}_1 = 420.22 + 51.14 = 471.36$.
- Estimated mean subject relative reading time: $\hat{\beta}_0 - \hat{\beta}_1 = 420.22 - 51.14 = 369.08$.

The difference in the means on the raw scale is 102 ms. The larger estimate based on the raw scale is less realistic, and we will see later that the large difference between the two conditions is driven by a few extreme, influential values.

3.4 From the paired t-test to the linear mixed model

One important point to notice is that the observed t-value of the paired t-test and the t-test printed out by the linear model don't match:

```
t.test(rawRT~condition,bysubj,paired=TRUE)$statistic
```

```
##     t
## 3.109
```

```
round(summary(m0)$coefficients,2)[,c(1:3)]
```

```
##                   Estimate Std. Error t value
## (Intercept)          471.4      31.13   15.14
## conditionsubjgap    -102.3      44.02   -2.32
```

This is because the linear model is equivalent to the unpaired (i.e., two sample) t-test:

```
summary(lm(rawRT~condition,bysubj))
```

```
##
## Call:
## lm(formula = rawRT ~ condition, data = bysubj)
##
## Residuals:
##    Min     1Q Median     3Q    Max
## -303.4 -116.4  -51.6   49.1  853.3
##
## Coefficients:
##                   Estimate Std. Error t value Pr(>|t|)
## (Intercept)          471.4       31.1   15.14   <2e-16
## conditionsubjgap    -102.3       44.0   -2.32    0.023
##
## Residual standard error: 202 on 82 degrees of freedom
## Multiple R-squared:  0.0618, Adjusted R-squared:  0.0503
## F-statistic:  5.4 on 1 and 82 DF,  p-value: 0.0226
```

```
round(t.test(rawRT~condition,bysubj,
             paired=FALSE)$statistic,2)
```

```
##    t
## 2.32
```

The paired t-test has an equivalent in the linear modeling framework: the linear mixed model. We turn next to this extension of

the simple linear model. The command corresponding to the paired t-test in the linear modeling framework is:

```
m0.lmer<-lmer(rawRT~condition+(1|subject),bysubj)
summary(m0.lmer)$coefficients
```

```
##                   Estimate Std. Error t value
## (Intercept)          471.4      31.13  15.143
## conditionsubjgap    -102.3      32.90  -3.109
```

To understand the connection between the paired t-test and the above command, it is necessary to consider how a paired t-test is assembled.

First, some background. If you have two random variables that have correlation ρ, the variance of the difference between the two random variables is:

$$Var(X_1 - X_2) = Var(X_1) + Var(X_2) - 2 \times Cov(X_1, X_2) \quad (3.18)$$

$Cov(X_1, X2)$ is the covariance between the two random variables and is defined as:

$$Cov(X_1, X_2) = \rho\sqrt{Var(X_1)}\sqrt{Var(X_2)} \quad (3.19)$$

You can find the proofs of the above assertions in books like Rice (1995).

As discussed earlier, a paired t-test is used when you have paired data from subject $i = 1, ..., n$ in two conditions, say conditions 1 and 2. Let's write the data as two vectors X_1, X_2. Because the pairs of data points are coming from the same subject, they are correlated with some correlation ρ. Assume that both conditions 1 and 2 have standard deviation σ.

To make this discussion concrete, let's generate some simulated

bivariate data that are correlated. Assume that $\sigma = 1$, $\rho = 0.5$, and that the data are balanced.

```
library(MASS)
samplesize<-12
mu <- c(.3, .2)
rho<-0.5
stddev<-1
Sigma <- matrix(stddev, nrow=2, ncol=2) + diag(2)
Sigma<-Sigma/2
Sigma
```

```
##      [,1] [,2]
## [1,]  1.0  0.5
## [2,]  0.5  1.0
```

```
## simulated data:
x <- mvrnorm(n=samplesize, mu=mu, Sigma=Sigma, empirical=TRUE)
head(x)
```

```
##         [,1]    [,2]
## [1,] -0.5361  0.1271
## [2,]  1.3814  1.3024
## [3,] -0.7434 -0.1521
## [4,]  2.3506  0.3507
## [5,] -1.2914 -1.4430
## [6,]  0.1007 -0.9218
```

```
n<-samplesize
x1<-x[,1]
x2<-x[,2]
x1
```

```
##  [1] -0.5361  1.3814 -0.7434  2.3506 -1.2914  0.1007
##  [7]  0.5587 -0.3438  0.5338 -0.0866  0.8119  0.8640
```

```
x2
```

```
##  [1]  0.1271  1.3024 -0.1521  0.3507 -1.4430 -0.9218
##  [7]  0.3375 -0.3714 -0.8318  1.5399  1.6450  0.8175
```

To carry out the paired t-test, we need to know the variance of $X_1 - X_2$ because the t-statistic will be:

$$t_{n-1} = \frac{X_1 - X_2}{\sqrt{Var(X_1 - X_2)/n}} \tag{3.20}$$

Now,

$$Var(X_1 - X_2) = \sigma^2 + \sigma^2 - 2\rho\sigma\sigma = 2\sigma^2(1-\rho) \tag{3.21}$$

Now let's compute the t-statistic using the above formula. Let the actual data vectors be x_1, x_2.

$$t_{n-1} = \frac{mean(x_1) - mean(x_2)}{\sqrt{Var(X_1 - X_2)/n}} \tag{3.22}$$

This simplifies to:

$$t_{n-1} = \frac{mean(x_1) - mean(x_2)}{\sqrt{2\sigma^2(1-\rho)/n}} \tag{3.23}$$

Now compare the paired t-test output and the by-hand calculation:

```
t.test(x1,x2,paired=TRUE)$statistic
```

```
##      t
## 0.3464
```

```
(mean(x1)-mean(x2))/sqrt((2*stddev^2*(1-rho))/n)
```

```
## [1] 0.3464
```

The linear mixed model we present next will fit exactly the same model as in the paired t-test above. To see this, suppose we have i subjects and $j = 1, 2$ conditions. For simplicity, assume that each subject sees each condition once (e.g., the by-subjects aggregated English relative clause data), so we have two data points from each subject. In other words, the data are paired.

Then, for condition 1, the dependent variable can be described by the equation:

$$y_{i1} = \beta_0 + u_i + \varepsilon_{i1}$$

Here, β_0 is the mean reading time, and ε is the usual residual error term. The interesting new term is u_i. This is the adjustment to the mean reading time for subject i. That is, if some subject is slower than average, u_i will be a positive number; if a subject is faster than average, then that subject's adjustment u_i will be negative in sign; and if a subject has exactly the same reading time as the mean for all subjects, then u_i for that subject will be 0.

Similarly, for condition 2, the dependent variable is described by the equation:

$$y_{i2} = \beta_0 + \delta + u_i + \varepsilon_{i2}$$

Here, δ is the additional time taken to process condition 2 (thus, this is the treatment contrast coding we saw earlier in this chapter).

If we subtract the equation for condition 2 from the equation for condition 1, the resulting equation is:

$$d_i = y_{i1} - y_{i2} = \delta + (\varepsilon_{i1} - \varepsilon_{i2})$$

The expectation of d_i is δ because the expectation of the ε terms is 0 (we set up the model such that $\varepsilon \sim Normal(0, \sigma)$).

Now, assuming that the error terms are correlated with correlation ρ, the result presented at the beginning of this section applies:

$$Var(y_{i1} - y_{i2}) = \sigma^2 + \sigma^2 - 2\rho\sigma^2 = 2\sigma^2(1-\rho) \tag{3.24}$$

The generative distribution for d_i, the pairwise differences in the two conditions, is

$$d \sim Normal(\delta, \sqrt{2\sigma^2(1-\rho)}) \tag{3.25}$$

But that is exactly the same standard deviation as the one used in the paired t-test.

So, the paired t-test will deliver exactly the same t-score as the above linear mixed model.

Let's check that this is true using our simulated data. In the code below, the term `(1|subj)` is the adjustment by subject to the intercepts—the term $u0_i$ above.

```
library(lme4)
dat<-data.frame(y=c(x1,x2),cond=rep(letters[1:2],each=n),subj=rep(1:n,2))
dat$cond<-factor(dat$cond)
contrasts(dat$cond)<-contr.sum(2)
contrasts(dat$cond)
```

```
##   [,1]
## a    1
## b   -1
```

```
summary(m<-lmer(y~cond+(1|subj),dat))
```

```
## Linear mixed model fit by REML ['lmerMod']
## Formula: y ~ cond + (1 | subj)
##    Data: dat
##
## REML criterion at convergence: 65.6
##
## Scaled residuals:
```

```
##     Min      1Q  Median      3Q     Max
## -1.0830 -0.7651  0.0838  0.3807  1.8623
##
## Random effects:
##  Groups   Name        Variance Std.Dev.
##  subj     (Intercept) 0.5      0.707
##  Residual             0.5      0.707
## Number of obs: 24, groups:  subj, 12
##
## Fixed effects:
##             Estimate Std. Error t value
## (Intercept)    0.250      0.250    1.00
## cond1          0.050      0.144    0.35
##
## Correlation of Fixed Effects:
##       (Intr)
## cond1 0.000
```

The t-statistic from the linear mixed model is exactly the same as that from the paired t-test.

With this as background, we are ready to look at linear mixed models in detail.

3.5 Linear mixed models

We return to our subject and object relative clause data from English (Grodner and Gibson, Expt 1). First we load the data as usual, define relative clause type as a sum coded predictor, and create a new column called `so` that represents the contrast coding (± 1 sum contrasts), and a column that holds log-transformed reading time.

```
gg05e1<-read.table("data/grodnergibsonE1crit.txt",header=TRUE)
```

```
gg05e1$so <- ifelse(gg05e1$condition=="objgap",1,-1)
gg05e1$logrt<-log(gg05e1$rawRT)
```

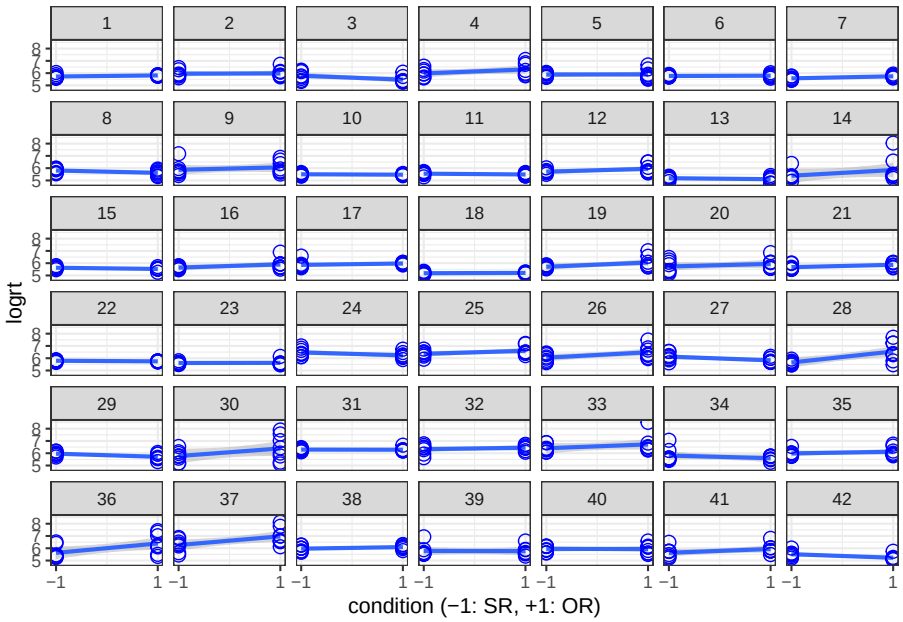
Recall that these data have multiple measurements from each subject for each condition:

```
t(xtabs(~subject+condition, gg05e1))
```

```
##           subject
## condition 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18
##   objgap  8 8 8 8 8 8 8 8 8  8  8  8  8  8  8  8  8  8
##   subjgap 8 8 8 8 8 8 8 8 8  8  8  8  8  8  8  8  8  8
##           subject
## condition 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33
##   objgap   8  8  8  8  8  8  8  8  8  8  8  8  8  8  8
##   subjgap  8  8  8  8  8  8  8  8  8  8  8  8  8  8  8
##           subject
## condition 34 35 36 37 38 39 40 41 42
##   objgap   8  8  8  8  8  8  8  8  8
##   subjgap  8  8  8  8  8  8  8  8  8
```

We can visualize the different responses of subjects:

```
## `geom_smooth()` using formula 'y ~ x'
```



It's clear that different subjects have different effects of the relative clause manipulation: some slopes are positive sloping, some are flat, and some are negatively sloping. There is between-subject variability in the relative clause effect.

Given these differences between subjects, you could fit a separate linear model for each subject, collect together the intercepts and slopes for each subject, and then check if the slopes are significantly different from zero. There is a function in the package `lme4` that computes separate linear models for each subject: *lmList*.

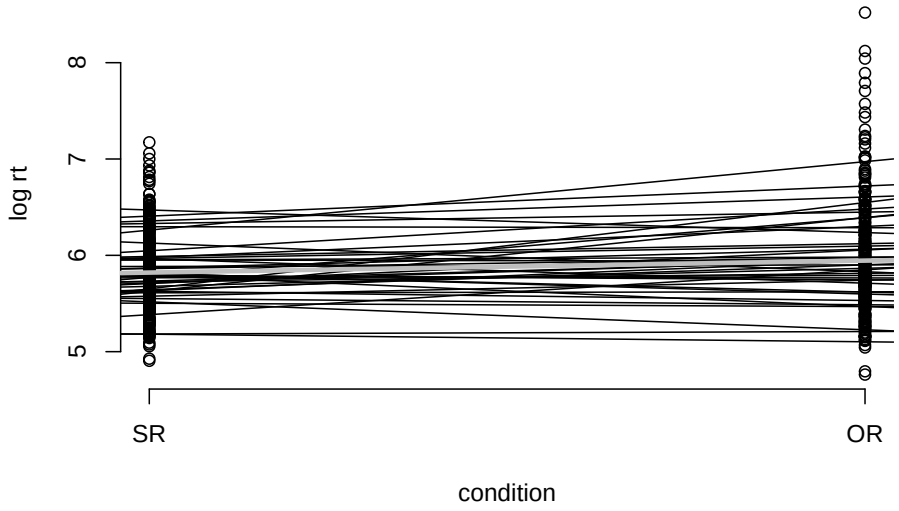
```
library(lme4)

lmlist.fm1<-lmList(logrt~so|subject,gg05e1)
```

One can extract the intercept and slope estimates for each subject. For example, for subject 1:

```
lmlist.fm1$`1`$coefficients
```

```
## (Intercept)          so
##     5.76962     0.04352
```

One can plot the individual lines for each subject, as well as the fit of a simple linear model m0 for all the data taken together; this will show how each subject deviates in intercept and slope from the model m0's intercept and slope.



To find out if there is an effect of relative clause type, we simply need to check whether the slopes of the individual subjects' fitted lines taken together are significantly different from zero. A one-sample t-test will achieve this:

```
t.test(coef(lmlist.fm1)[2])
```

```
##
##  One Sample t-test
##
## data:  coef(lmlist.fm1)[2]
## t = 2.8, df = 41, p-value = 0.008
## alternative hypothesis: true mean is not equal to 0
## 95 percent confidence interval:
##  0.01745 0.10658
## sample estimates:
```

```
## mean of x
##   0.06202
```

The above test is *exactly* the same as the paired t-test and the varying intercepts linear mixed model that we fit in the last chapter using the by-subject aggregated data:

```
bysubj<-aggregate(log(rawRT)~subject+condition,
                  mean,data=gg05e1)

colnames(bysubj)[3]<-"logrt"

t.test(logrt~condition,bysubj,paired=TRUE)$statistic
```

```
##    t
## 2.81
```

```
## compare with linear mixed model:
summary(lmer(logrt~condition+(1|subject),
             bysubj))$coefficients[2,]
```

```
##   Estimate Std. Error    t value
##   -0.12403    0.04414   -2.81021
```

The above lmList model we just fit is called *repeated measures regression*. We now look at how to model unaggregated data using the linear mixed model. Incidentally, this repeated measures regression model is now largely of historical interest, and useful only for understanding the linear mixed model, which is the modern standard approach.

We turn next to three main types of linear mixed model; other variants will be introduced in later chapters.

3.5.1 Model type 1: Varying intercepts

The *linear mixed model* does something related to the above by-subject fits, but with some crucial twists, as we see below. In the model shown below, the statement

$$(1 \mid \text{subject}) \tag{3.26}$$

adjusts the grand mean estimates of the intercept by a term (a number) for each subject.

```
m0.lmer<-lmer(logrt~so+(1|subject),gg05e1)
```

Notice that we did not aggregate the data.

Here is the abbreviated output:

```
Random effects:
 Groups   Name        Variance Std.Dev.
 subject  (Intercept) 0.09983  0.3160
 Residual             0.14618  0.3823
Number of obs: 672, groups:  subject, 42

Fixed effects:
            Estimate Std. Error t value
(Intercept)  5.88306    0.05094 115.497
so           0.06202    0.01475   4.205
```

One thing to notice in the present example is that the coefficients (intercept and slope) of the fixed effects of the above model are identical to those in the linear model `m0` above. What is different between the linear model and the linear mixed model is the standard error. In the latter, the standard error is determined by more than one source of variance, as we explain below.

The intercept adjustments for each subject can be viewed by typing:

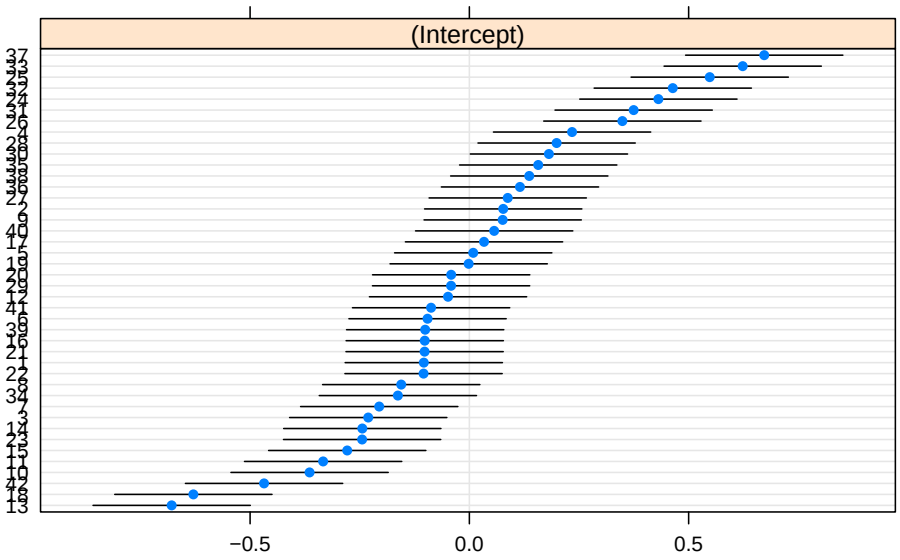
```
## first 10 subjects' intercept adjustments:
ranef(m0.lmer)$subject[,1][1:10]
```

```
##  [1] -0.103928  0.077195 -0.230621  0.234198  0.008828
##  [6] -0.095363 -0.205571 -0.155371  0.075944 -0.364367
```

Here is another way to summarize the adjustments to the grand mean intercept by subject. The error bars represent 95% confidence intervals.

```
library(lattice)
print(dotplot(ranef(m0.lmer,condVar=TRUE)))
```

```
## $subject
```



3.5.2 The formal statement of the varying intercepts model

The model `m0.lmer` above prints out the following type of linear model. i indexes subject, and j indexes items.

Once we know the subject id and the item id, we know which subject saw which condition:

```
subset(gg05e1,subject==1 & item == 1)
```

```
##   subject item condition rawRT so logrt
## 6       1    1    objgap   320  1 5.768
```

The mathematical form of the linear mixed model is:

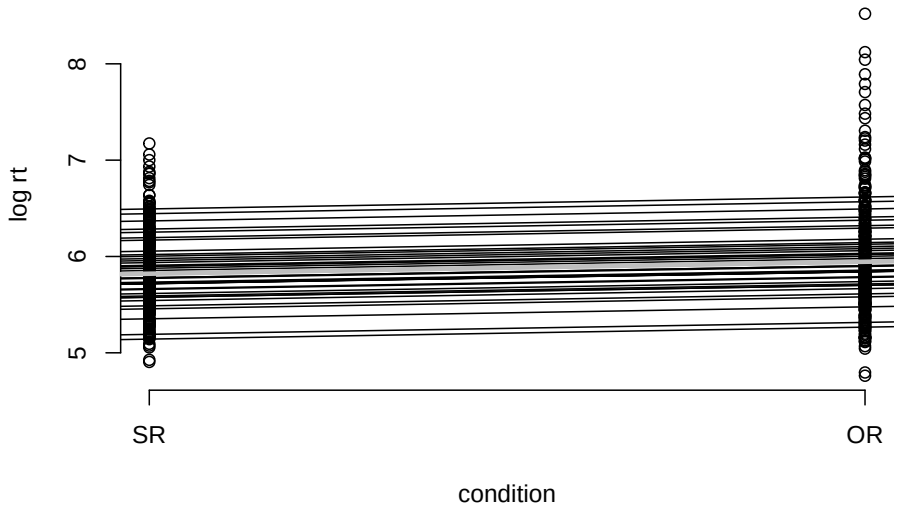
$$y_{ij} = \beta_0 + u_{0i} + \beta_1 \times so_{ij} + \varepsilon_{ij} \tag{3.27}$$

The *only* new thing here beyond the linear model we saw earlier is the by-subject adjustment to the intercept. These by-subject adjustments to the intercept u_{0i} are assumed by lmer to come from a normal distribution centered around 0:

$$u_{0i} \sim Normal(0, \sigma_{u0}) \tag{3.28}$$

The ordinary linear model m0 has one intercept β_0 for all subjects, whereas this linear mixed model with varying intercepts `m0.lmer` has a different intercept $(\beta_0 + u_{0i})$ for each subject i.

We can visualize the adjustments for each subject to the intercepts as shown below.



An important point is that in this model there are two variance components or sources of variance (cf. the linear model, which had only one):

- $u_0 \sim Normal(0, \sigma_{u0})$
- $\varepsilon \sim Normal(0, \sigma)$

These two standard deviations determine the standard error of the β_1 slope parameter.

3.5.3 Model type 2: Varying intercepts and slopes, without a correlation

Unlike the figure associated with the `lmlist.fm1` model above, which also involves fitting separate models for each subject, the model `m0.lmer` assumes *different intercepts* for each subject *but the same slope.*

We can choose to fit different intercepts as well as different slopes for each subject. To achieve this, assume now that each subject's slope is also adjusted:

$$y_{ij} = \beta_0 + u_{0i} + (\beta_1 + u_{1i}) \times so_{ij} + \varepsilon_{ij} \tag{3.29}$$

That is, we additionally assume that $u_{1i} \sim Normal(0, \sigma_{u1})$.

The `lmer` notation for fitting separate intercepts and slopes is `(1+so||subject)`. We will just explain what the double vertical bars represent.

```
m1.lmer<-lmer(logrt~so+(1+so||subject),gg05e1)
```

The output of this model will now show that there are not two but three sources of variability. These are:

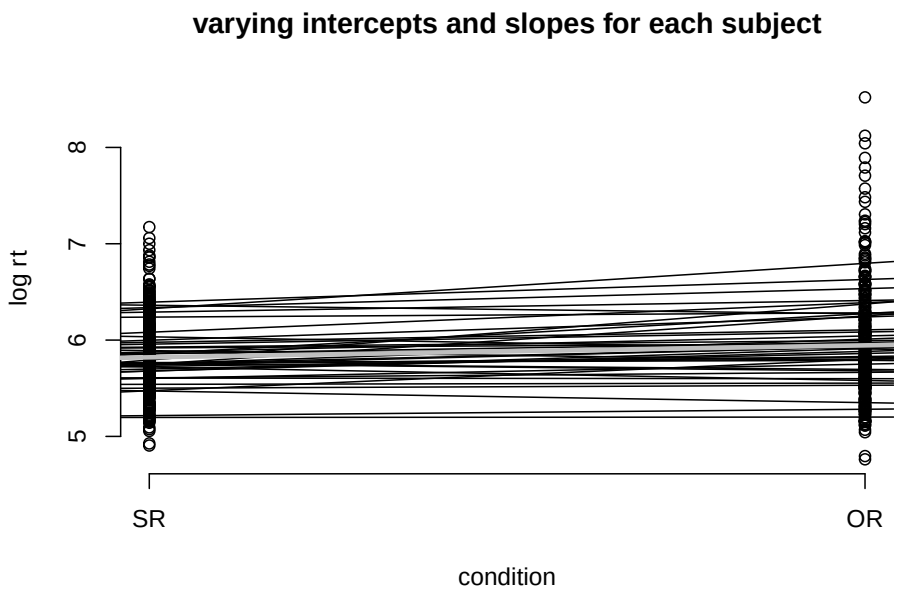
- $u_0 \sim Normal(0, \sigma_{u0})$
- $u_1 \sim Normal(0, \sigma_{u1})$
- $\varepsilon \sim Normal(0, \sigma)$

In particular, the model estimates the following standard deviations:

- $\hat{\sigma}_{u0} = 0.317$
- $\hat{\sigma}_{u1} = 0.110$
- $\hat{\sigma} = 0.365$.

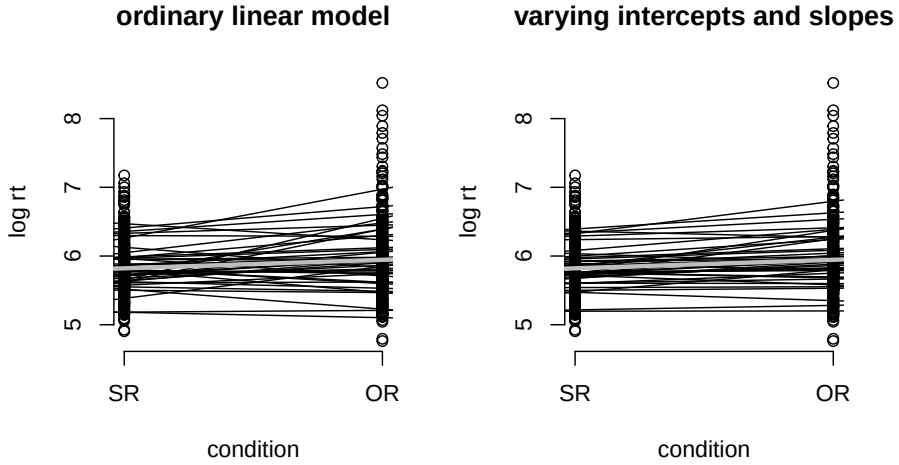
```
Random effects:
 Groups    Name        Variance Std.Dev.
 subject   (Intercept) 0.1006   0.317
 subject.1 so          0.0121   0.110
 Residual              0.1336   0.365
Number of obs: 672, groups:  subject, 42

Fixed effects:
            Estimate Std. Error t value
(Intercept)   5.8831     0.0509  115.50
so            0.0620     0.0221    2.81
```

These fits for each subject are visualized below (the gray line shows the model with a single intercept and slope, i.e., our old model `m0`):



3.5.3.1 Comparing lmList model with the varying intercepts model

Compare this model with the `lmlist.fm1` model we fitted earlier:



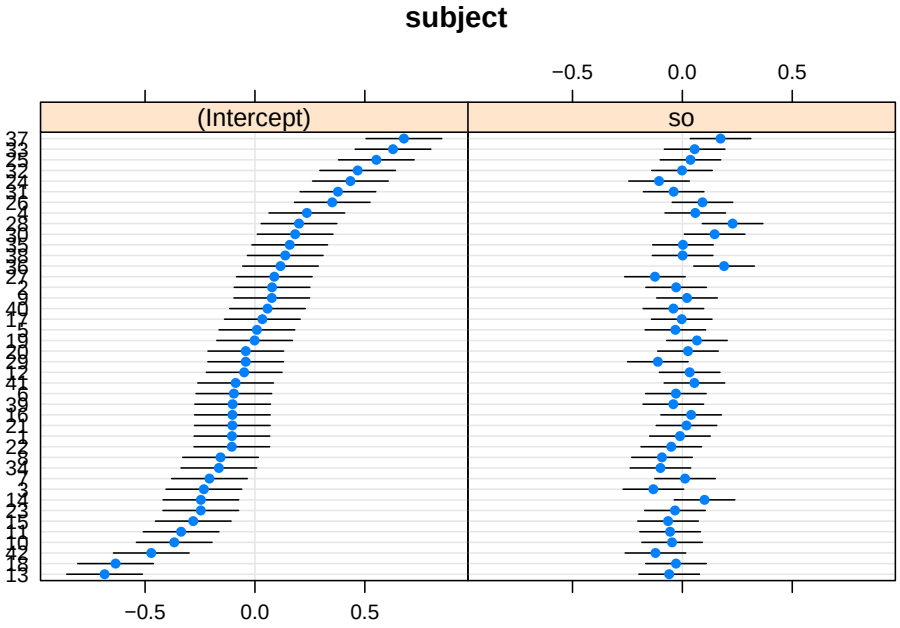
What is striking is that each subject's estimated best fit line is "smoothed out" compared to the lmList fits. This aspect of the linear mixed model is called shrinkage; we return to this point shortly.

3.5.3.2 Visualizing random effects

As before, it is instructive to visualize the by-subjects adjustments to the intercept and slope:

```
print(dotplot(ranef(m1.lmer,condVar=TRUE)))
```

```
## $subject
```



What this is showing is wide variability in the mean reading times between subjects, but very little variation in the slope between subjects.

3.5.3.3 The formal statement of varying intercepts and varying slopes linear mixed model

Here is the full statement of the varying intercept and slopes model. Again, i indexes subjects, j items.

$$y_{ij} = \beta_0 + u_{0i} + (\beta_1 + u_{1i}) \times so_{ij} + \varepsilon_{ij} \tag{3.30}$$

There are now three variance components:

- $u_0 \sim Normal(0, \sigma_{u0})$
- $u_1 \sim Normal(0, \sigma_{u1})$
- $\varepsilon \sim Normal(0, \sigma)$

3.5.3.4 Crossed random effects for subjects and for items

The varying intercepts and slopes model doesn't capture all the sources of variance yet. The items also contribute sources of variance: just like subjects, items may also vary in their reading times or in the extent to which the reading times are impacted by condition. I other words, they might have different intercepts and slopes.

Notice that in this design (as in many designs in psycholinguistics or linguistic research) each subject sees all the items. In such cases, we say that subjects and items are crossed.

```
head(xtabs(~subject+item,gg05e1))
```

```
##         item
## subject 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
##       1 1 1 1 1 1 1 1 1 1  1  1  1  1  1  1  1
##       2 1 1 1 1 1 1 1 1 1  1  1  1  1  1  1  1
##       3 1 1 1 1 1 1 1 1 1  1  1  1  1  1  1  1
##       4 1 1 1 1 1 1 1 1 1  1  1  1  1  1  1  1
##       5 1 1 1 1 1 1 1 1 1  1  1  1  1  1  1  1
##       6 1 1 1 1 1 1 1 1 1  1  1  1  1  1  1  1
```

Linear mixed model with crossed subject and items random effects can be defined with the following syntax:

```
m2.lmer<-lmer(logrt~so+(1+so||subject)+
(1+so||item),gg05e1)
```

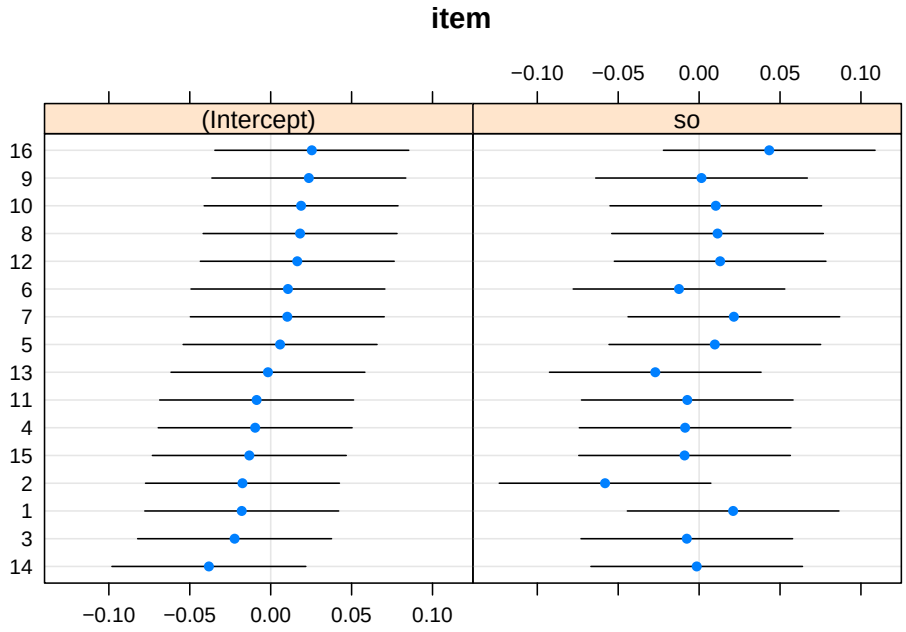
Analogously to the preceding example, now there are five variance components:

```
Random effects:
 Groups    Name        Variance Std.Dev.
```

```
 subject    (Intercept) 0.10090  0.3177
 subject.1  so          0.01224  0.1106
 item       (Intercept) 0.00127  0.0356
 item.1     so          0.00162  0.0402
 Residual               0.13063  0.3614
Number of obs: 672, groups:  subject, 42; item, 16

Fixed effects:
             Estimate Std. Error t value
(Intercept)   5.8831     0.0517  113.72
so            0.0620     0.0242    2.56
```

The item intercept and slope adjustments can be visualized as well. Notice that there is a lot less item-level variation; this is often the case in planned experiments in sentence processing, where the experimental items are carefully constructed to vary as little as possible.

```
print(dotplot(ranef(m2.lmer,condVar=TRUE))$item)
```



In the above models, there is an assumption that there is no cor-

relation between the intercept and slope adjustments by subject, and no correlation between the intercept and slope adjustments by item. It is possible that the intercept and slope adjustments are in fact correlated. We turn to this model next.

3.5.4 Model type 3: Varying intercepts and varying slopes, with correlation

A correlation can be introduced between the intercept and slope adjustments by using a single vertical bar instead of two vertical bars in the random effects structure:

```
m3.lmer<-lmer(logrt~so+(1+so|subject)+(1+so|item),
              gg05e1)
```

```
## boundary (singular) fit: see ?isSingular
```

To understand what this model is doing, we have to recall what a bivariate/multivariate distribution is.

```
Random effects:
 Groups   Name        Variance Std.Dev. Corr
 subject  (Intercept) 0.10103  0.3178
          so          0.01228  0.1108   0.58
 item     (Intercept) 0.00172  0.0415
          so          0.00196  0.0443   1.00 <= degeneracy
 Residual             0.12984  0.3603
Number of obs: 672, groups:  subject, 42; item, 16

Fixed effects:
            Estimate Std. Error t value
(Intercept)   5.8831     0.0520  113.09
so            0.0620     0.0247    2.51
```

The correlations (0.58 and 1.00) you see in the model output below are the correlations between the varying intercepts and slopes for subjects and for items. Notice that the variance covariance matrix for items is degenerate: its correlation is 1. This matrix cannot be inverted.

When the correlation is +1 or -1 or near these numbers, this means that the optimizer in lme4 is unable to estimate the correlation parameter, usually due to there not being enough data. If you are in such a situation, you are better off not trying to estimate this parameter with the data you have, and instead fitting one of the simpler models. We will return to this point when discussing model selection. For further discussion, see Barr et al. (2013), Bates et al. (2015), and Matuschek et al. (2017).

3.5.4.1 Formal statement of varying intercepts and varying slopes linear mixed model with correlation

As usual, i indexes subjects, j items. The vector `so` is the sum-coded factor levels: +1 for object relatives and -1 for subject relatives. The only new thing in this model is the item-level effects, and the specification of the variance-covariance matrix for subjects and items, in order to include the correlation parameters.

$$y_{ij} = \alpha + u_{0i} + w_{0j} + (\beta + u_{1i} + w_{1j}) \times so_{ij} + \varepsilon_{ij} \tag{3.31}$$

where $\varepsilon_{ij} \sim Normal(0, \sigma)$ and

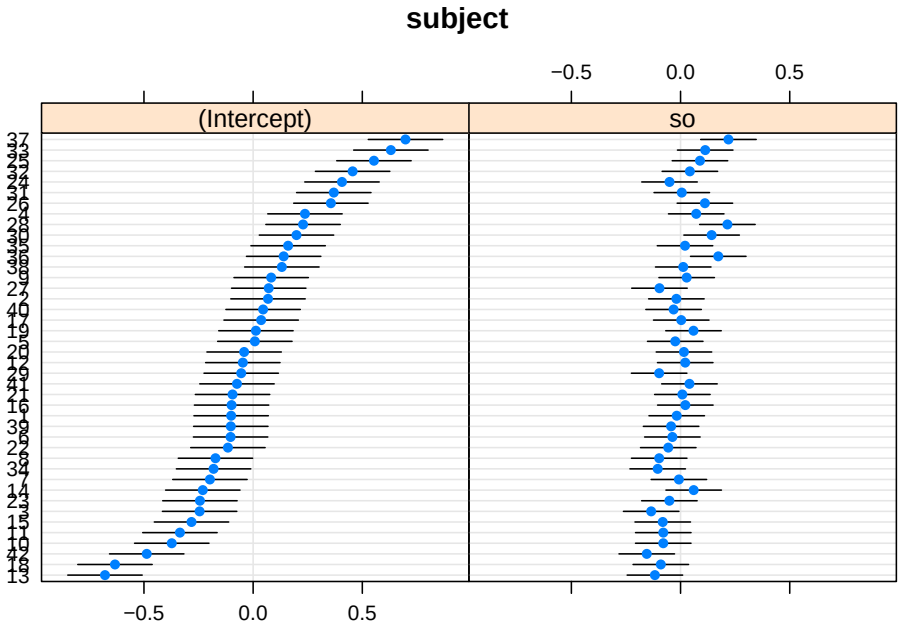
$$\Sigma_u = \begin{pmatrix} \sigma^2_{u0} & \rho_u \sigma_{u0} \sigma_{u1} \\ \rho_u \sigma_{u0} \sigma_{u1} & \sigma^2_{u1} \end{pmatrix} \quad \Sigma_w = \begin{pmatrix} \sigma^2_{w0} & \rho_w \sigma_{w0} \sigma_{w1} \\ \rho_w \sigma_{w0} \sigma_{w1} & \sigma^2_{w1} \end{pmatrix} \tag{3.32}$$

$$\begin{pmatrix} u_0 \\ u_1 \end{pmatrix} \sim \mathcal{N}\left(\begin{pmatrix} 0 \\ 0 \end{pmatrix}, \Sigma_u \right), \quad \begin{pmatrix} w_0 \\ w_1 \end{pmatrix} \sim \mathcal{N}\left(\begin{pmatrix} 0 \\ 0 \end{pmatrix}, \Sigma_w \right) \tag{3.33}$$

3.5.4.2 Visualizing the random effects

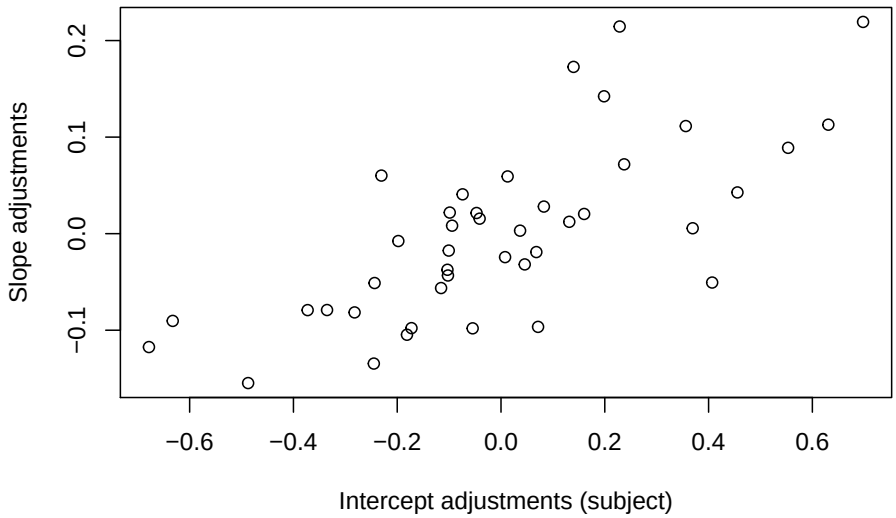
One can visualize the correlation between intercepts and slopes by subjects. The positive correlation of 0.58 between subject intercept and slope adjustments implies that slower subjects show larger effects. However, the dotplot below doesn't show a convincing indication that such a correlation exists:

```
print(dotplot(ranef(m3.lmer,condVar=TRUE))$subject)
```



The correlation pattern is easier to see if we plot the slope adjustments against the intercept adjustments.

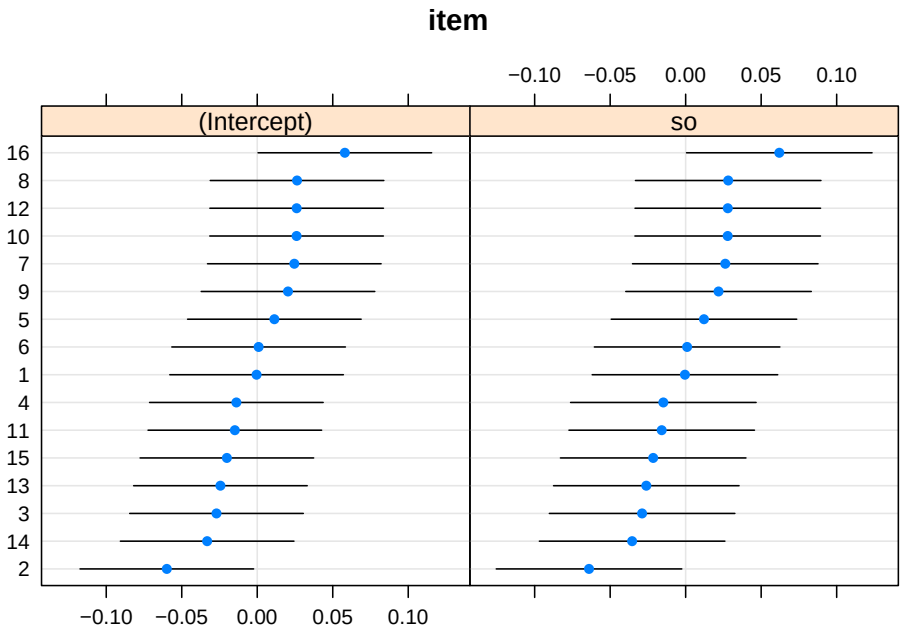
```
plot(ranef(m3.lmer)$subject[,1],ranef(m3.lmer)$subject[,2],
xlab="Intercept adjustments (subject)",
ylab="Slope adjustments")
```



When we talk about hypothesis testing, we will look at what inferences we can draw from this correlation.

The dotplot showing the item-level effects shows a perfect correlation between intercept and slope adjustments, but as mentioned above these are from a degenerate variance covariance matrix and not meaningful.

```
print(dotplot(ranef(m3.lmer,condVar=TRUE))$item)
```



3.6 Shrinkage in linear mixed models

The estimate of the effect for each participant computed from a linear mixed model is "pushed" towards the grand mean effect compared to when we fit a separate linear model to each subject's data. This is called "shrinkage" in linear mixed models. We say that the individual-level estimates are shrunk towards the mean slope. The less data we have from a given subject, the greater the shrinkage.

3.6.0.1 Shrinkage in action: when data are missing

The importance and value of shrinkage becomes clear once we simulate a situation where there is some missing data. Missingness can happen in experiments, either due to lost measurements (arising from computer error or programming errors), or some other reason. To see what happens when we have missing data, let's randomly delete some data from one subject. We will randomly delete 50% of subject 37's data:

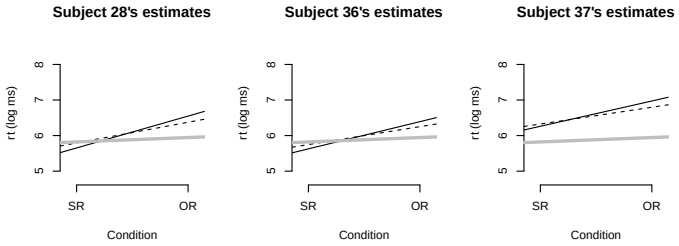


FIGURE 3.1: The figures show linear model fits (the grand mean estimates) for three subjects; shown are the simple linear model fit on all the data (gray line), the lmList model fit to the individual subject's data (black line), and the linear mixed model fit (the broken line). In all three subjects' models, the linear mixed model estimates are shrunk towards the grand mean (gray line) estimates.

```
set.seed(4321)
## choose some data randomly to remove:
rand<-rbinom(1,n=16,prob=0.5)
```

Here are subject 37's reading times (16 data points):

```
gg05e1[which(gg05e1$subject==37),]$rawRT
```

```
##  [1]  770  536  686  578  457  487 2419  884 3365  233
## [11]  715  671 1104  281 1081  971
```

Now, we randomly delete half the data:

```
gg05e1$deletedRT<-gg05e1$rawRT
gg05e1[which(gg05e1$subject==37),]$deletedRT<-

  ifelse(rand,NA,
         gg05e1[which(gg05e1$subject==37),]$rawRT)
```

Now subject 37's estimates are going to be pretty wild, because

they are based on much less data (even one extreme value can strongly influence the mean):

```
subset(gg05e1,subject==37)$deletedRT
```

```
##  [1]  770   NA  686  578   NA   NA   NA   NA 3365  233
## [11]   NA  671 1104   NA   NA  971
```

```
## [1] 6.617
```

```
## [1] 0.3554
```

```
## [1] 6.688
```

```
## [1] 0.3884
```

Now fit the hierarchical model and examine subject 37's estimates on undeleted vs deleted data:

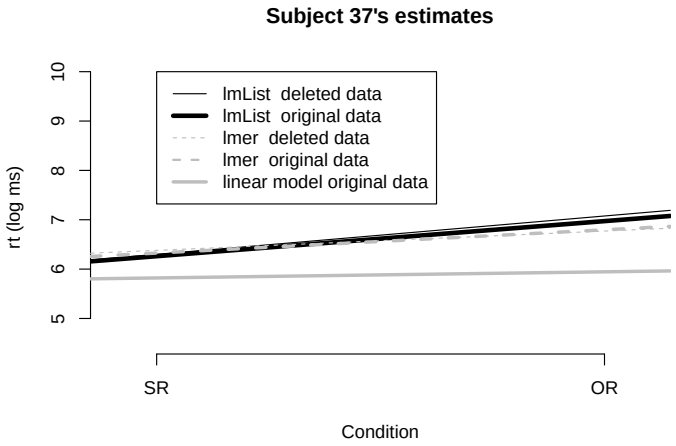


FIGURE 3.2: The figure shows linear model fits (the grand mean estimates) for subject 37. When using lmList, deleting data leads to very different estimates; but using lmer, deleting half the data from this subject hardly affects the individual subject's estimates.

What we see here is that the estimates from the hierarchical model are barely affected by the missingness, but the estimates from the lmList model are heavily affected. This means that linear mixed models will give you more robust estimates (think Type M error!)

compared to models which fit data sepaarately for each subject. This property of shrinkage is one reason why linear mixed models are so important in cognitive science.

3.7 Summary

3.8 Exercises

Download the data-set E1_data.csv. Then run the following commands to load the `lme4` library and to set up your data for analysis:

```
library(lme4)

## load data:
dat<-read.csv("data/E1_data.csv",header=TRUE)
## convert RT to milliseconds:
dat$RT<-dat$RT*1000
## choose critical region:
word_n<-4
## subset critical data:
crit<-subset(dat,Position==word_n)
```

The data consist of a repeated measures experiment comparing two conditions which are labeled Type 1 and Type 2. The column Sub refers to subject id, and the column ID refers to item id. RT refers to reading time in seconds (we have converted it above to milliseconds); NA is missing data. You can ignore the other columns. This is a standard Latin square design. We will work with the data frame `crit` below.

3.8.1 By-subjects t-test

Using RT as a dependent variable, carry out the appropriate by-subjects t-test to evaluate the null hypothesis that there is no

difference between the conditions labeled Type 1 and 2. Write down all the R commands needed to do the appropriate t-test, and the resulting t-value and p-value. State whether we can reject the null hypothesis given the results of the t-test; explain why.

3.8.2 Fitting a linear mixed model

Now, using the data-frame called `crit` above, fit a linear mixed model (called M0). Recode the column called Type as sum contrasts.

Assume varying intercepts for subjects and varying intercepts for items (varying intercepts are sometimes called random intercepts). Write down the linear mixed models command, and write down the fixed-effects estimates (intercept and slope) along with their standard errors. State whether we can reject the null hypothesis given the results of the t-value shown in the linear mixed model output; explain why.

3.8.3 t-test vs. linear mixed model

Why do the results of the t-test and the linear mixed model M0 differ?

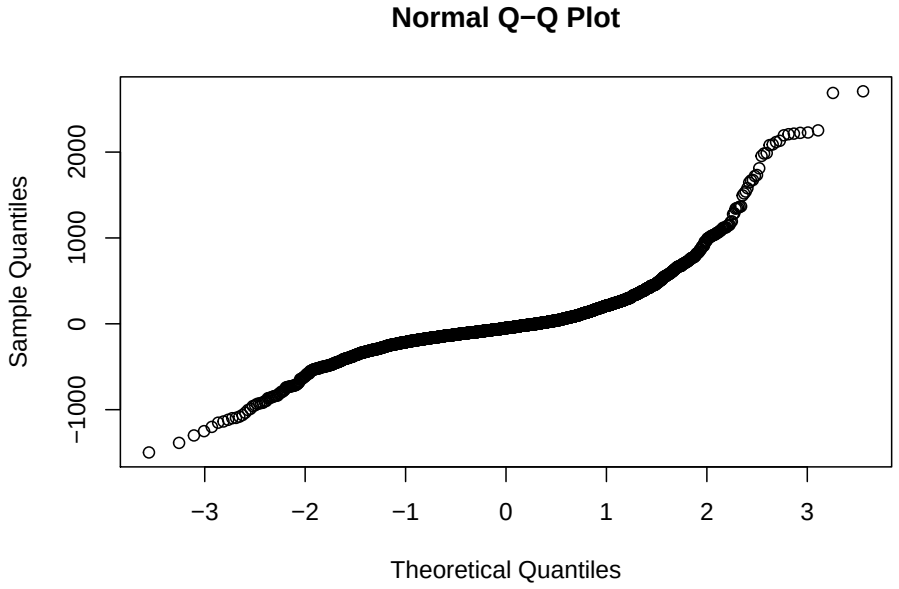
3.8.4 Power calculation using power.t.test

The researcher wants to achieve 80% statistical power in a future study. Based on the available data above, she determines that the standard error (note: not the standard deviation!) of the difference in means between the conditions Type 1 and Type 2 is 21. She has reason to believe that the true difference in means is 30 ms. What is the number of participants (to the nearest whole number) needed to achieve approximately 80% power? Use the power.t.test function to compute your answer. Write down the `power.t.test` function specification you used, as well as the number of participants needed, based on the output of the power.t.test function.

3.8.5 Residuals

The plot below shows the distribution of the residuals from model M0 plotted against the standard normal distribution with mean 0 and standard deviation 1. Explain what the plot tells us about one of the model assumptions of the linear mixed model M0 that you fit earlier.

(You can ignore the numbers below the plot.)



3.8.6 Understanding contrast coding

Using only your estimates of the intercept and the slope in model M0's fixed effects output, write down the mean of the condition labeled Type 1 in the data, and the mean of the condition labeled Type 2.

3.8.7 Understanding the fixed-effects output

Suppose that the model M0's output for the fixed effects analysis were as follows. SO is a sum-coded contrast specification for the conditions in the column labeled Type.

```
results
```

```
##             Estimate Std. Error t value
## (Intercept)   686.01      47.54   14.43
## SO             18.94         NA    2.00
```

What is the value of the standard error of the slope (SO), which is labeled NA above?

3.8.8 Understanding the null hypothesis test

A researcher fits a linear mixed model to compare the reading times between two conditions (a) and (b), just like in the above study. Her hypothesis is that the mean for condition (a) is larger than the mean for (b). She observes that condition a has sample mean 500 ms, and condition (b) has sample mean 450 ms. She also establishes from the linear mixed model that the t-value is 1.94. The approximate p-value associated with this t-value is 0.052. Answer the following: (A) Do we have evidence against the null hypothesis and (B) do we have evidence for the particular research hypothesis that the researcher has?

The researcher runs the same analysis as above on a new data-set that has the same design as above, and now gets a p-value of 0.001. Now she has stronger evidence than in the above case where the p-value was 0.052. What does she have stronger evidence for?

4

Hypothesis testing using the likelihood ratio test

We started the book with the one-sample t-test. There, we had the following procedure:

- Given independent and identically distributed data y, define a null hypothesis: $H_0 : \mu = \mu_0$
- Compute the sample mean $\bar{y}$ and the standard error SE
- Reject the null hypothesis if the absolute value of $\bar{y}/SE$ is larger than 2.

Here, we turn to a closely related test: the *likelihood ratio test statistic*.

4.1 The likelihood ratio test: The theory

Suppose that $X_1, \ldots, X_n$ are independent and normally distributed with mean μ and standard deviation σ (assume for simplicity that σ is known).

Let the null hypothesis be $H_0 : \mu = \mu_0$ and the alternative be $H_1 : \mu \neq \mu_0$. Here, μ_0 is a number, such as 0.

The likelihood of the data y can be computed under the null model, in which $\mu = \mu_0$, and under the alternative model, in which μ has some specific alternative value. To make this concrete, imagine 10 data points being generated from a Normal(0,1).

```
y<-rnorm(10)
```

We can compute the joint likelihood under a null hypothesis that $\mu = 0$:

```
likNULL<-prod(dnorm(y,mean=0,sd=1))
likNULL
```

```
## [1] 9.151e-06
```

On the log scale, we would need to add the log likelihoods of each data point:

```
loglikNULL<-sum(dnorm(y,mean=0,sd=1,log=TRUE))
loglikNULL
```

```
## [1] -11.6
```

Similarly, we can compute the log likelihood with μ equal to the maximum likelihood estimate of μ, the sample mean.

```
loglikALT<-sum(dnorm(y,mean=mean(y),sd=1,log=TRUE))
loglikALT
```

```
## [1] -11.59
```

Essentially, the likelihood ratio test compares the ratio of likelihoods of the two models; on the log scale, the difference in log likelihood is taken. This is because a ratio on the log scale becomes a difference. The likelihood ratio test then chooses the model with the higher log likelihood, provided that the higher likelihood is high enough (we will just make this more precise).

One can specify the test in general terms as follows. Suppose that the likelihood is with respect to some parameter θ. We can evaluate the likelihood at μ_0, the null hypothesis value of the parameter, and evaluate the likelihood using the maximum likelihood estimate

$\hat{\theta}$ of the parameter, as we did above. The likelihood ratio can then be written as follows:

$$\Lambda = \frac{max_{\theta \in \omega_0}(lik(\theta))}{max_{\theta \in \omega_1)}(lik(\theta))} \tag{4.1}$$

where, $\omega_0 = \{\mu_0\}$ and $\omega_1 = \{\forall \mu \mid \mu \neq \mu_0\}$. The function max just selects the maximum value of any choices of parameter values; in the case of the null hypothesis there is only one value, μ_0. In the case of the alternative model, the maximum likelihood estimate $\hat{\theta}$ is the maximum value.

Now, assuming that the data are coming from a normal distribution, the numerator of the likelihood ratio statistic is:

$$lik(\theta = \mu_0) = \frac{1}{(\sigma\sqrt{2\pi})^n} exp\left(-\frac{1}{2\sigma^2}\sum_{i=1}^{n}(X_i - \mu_0)^2\right) \tag{4.2}$$

For the denominator, the MLE $\bar{X}$ is taken as μ:

$$lik(\theta = \bar{X}) = \frac{1}{(\sigma\sqrt{2\pi})^n} exp\left(-\frac{1}{2\sigma^2}\sum_{i=1}^{n}(X_i - \bar{X})^2\right) \tag{4.3}$$

The likelihood ratio statistic is then:

$$\Lambda = \frac{lik(\theta = \mu_0)}{lik(\theta = \bar{X})} = \frac{\frac{1}{(\sigma\sqrt{2\pi})^n} exp\left(-\frac{1}{2\sigma^2}\sum_{i=1}^{n}(X_i - \mu_0)^2\right)}{\frac{1}{(\sigma\sqrt{2\pi})^n} exp\left(-\frac{1}{2\sigma^2}\sum_{i=1}^{n}(X_i - \bar{X})^2\right)} \tag{4.4}$$

Canceling out common terms:

$$\Lambda = \frac{exp\left(-\frac{1}{2\sigma^2}\sum_{i=1}^{n}(X_i - \mu_0)^2\right)}{exp\left(-\frac{1}{2\sigma^2}\sum_{i=1}^{n}(X_i - \bar{X})^2\right)} \tag{4.5}$$

Taking logs:

$$\begin{aligned}\log\Lambda &= \left(-\frac{1}{2\sigma^2}\sum_{i=1}^{n}(X_i-\mu_0)^2\right) - \left(-\frac{1}{2\sigma^2}\sum_{i=1}^{n}(X_i-\bar{X})^2\right) \\ &= -\frac{1}{2\sigma^2}\left(\sum_{i=1}^{n}(X_i-\mu_0)^2 - \sum_{i=1}^{n}(X_i-\bar{X})^2\right)\end{aligned} \tag{4.6}$$

Now, it is a standard algebraic trick to rewrite $\sum_{i=1}^{n}(X_i-\mu_0)^2$ as a sum of two terms:

$$\sum_{i=1}^{n}(X_i-\mu_0)^2 = \sum_{i=1}^{n}(X_i-\bar{X})^2 + n(\bar{X}-\mu_0)^2 \tag{4.7}$$

If we rearrange terms, we obtain:

$$\sum_{i=1}^{n}(X_i-\mu_0)^2 - \sum_{i=1}^{n}(X_i-\bar{X})^2 = n(\bar{X}-\mu_0)^2 \tag{4.8}$$

Now, we just established above that $\log\Lambda$ is:

$$\log\Lambda = -\frac{1}{2\sigma^2}\left(\sum_{i=1}^{n}(X_i-\mu_0)^2 - \sum_{i=1}^{n}(X_i-\bar{X})^2\right) \tag{4.9}$$

Consider the term in the brackets:

$$(\sum_{i=1}^{n}(X_i-\mu_0)^2 - \sum_{i=1}^{n}(X_i-\bar{X})^2) \tag{4.10}$$

This can be rewritten as:

$$n(\bar{X}-\mu_0)^2 \tag{4.11}$$

Rewriting in this way gives us:

$$\log \Lambda = -\frac{1}{2\sigma^2} n(\bar{X} - \mu_0)^2 \tag{4.12}$$

Rearranging terms:

$$-2 \log \Lambda = \frac{n(\bar{X} - \mu_0)^2}{\sigma^2} \tag{4.13}$$

Or even more transparently:

$$-2 \log \Lambda = \frac{(\bar{X} - \mu_0)^2}{\frac{\sigma^2}{n}} \tag{4.14}$$

This should remind you of the t-test! Basically, just like in the t-test, what this is saying is that we reject the null when $\mid \bar{X} - \mu_0 \mid$, or negative two times the difference in log likelihood, is large!

Now we will define what it means for $-2 \log \Lambda$ to be large. There is a theorem in statistics that states that for large n, the distribution of $-2 \log \Lambda$ approaches the chi-squared distribution, with degrees of freedom corresponding to the difference in the number of parameters between the two models being compared.

We will define the *likelihood ratio test statistic*, call it LRT, as follows. Here, $Lik(\theta)$ refers to the likelihood given some value θ for the parameter, and $logLik(\theta)$ refers to the log likelihood.

$$\begin{aligned} LRT &= -2 \times (Lik(\theta_0)/Lik(\theta_1)) \\ \log LRT &= -2 \times \{logLik(\theta_0) - logLik(\theta_1)\} \end{aligned} \tag{4.15}$$

where θ_1 and θ_0 are the estimates of θ under the alternative and null hypotheses, respectively. The likelihood ratio test rejects H_0 if $\log LRT$ is sufficiently large. As the sample size approaches infinity, $\log LRT$ approaches the chi-squared distribution:

$$\log LRT \rightarrow \chi^2_r \text{ as } n \rightarrow \infty \tag{4.16}$$

Here, r is called the degrees of freedom and is the difference in the number of parameters under the null and alternative hypotheses.

The above result is called *Wilks' theorem*. The proof of Wilks' theorem is fairly involved but you can find it in Lehmann's textbook *Testing Statistical Hypotheses*.

Note that sometimes you will see the form:

$$\log LRT = 2\{logLik(\theta_1) - logLik(\theta_0)\} \tag{4.17}$$

It should be clear that both statements are saying the same thing; in the second case, we are just subtracting the null hypothesis log likelihood from the alternative hypothesis log likelihood, so the negative sign disappears.

That's the theory. Let's see how the likelihood ratio test works for (a) simulated data, and (b) our running example, the English relative clause data from Grodner and Gibson (2005).

4.2 A practical example using simulated data

A practical example will make the usage of this test clear. Let's just simulate data from a linear model:

```
x<-1:10
y<- 10 + 20*x+rnorm(10,sd=10)
```

Here, the null hypothesis that the slope is 0 is false (it has value 20). Now, we fit a null hypothesis model, without a slope:

```
## null hypothesis model:
m0<-lm(y~1)
```

We will compare this model's log likelihood with that of the alternative model, which includes an estimate of the slope:

```
## alternative hypothesis model:
m1<-lm(y~x)
```

The difference in log likelihood, multiplied with -2, is:

```
LogLRT<- -2*(logLik(m0)-logLik(m1))
## observed value:
LogLRT[1]
```

```
## [1] 34.49
```

The difference in the number of parameters in the two models is one, so $\log LRT$ has the distribution χ^2_1. Is the observed value 34.49 unexpected under this distribution? We can calculate the probability of obtaining the likelihood ratio statistic we observed above, or a value more extreme, given the χ^2_1 distribution.

```
pchisq(LogLRT[1],df=1,lower.tail=FALSE)
```

```
## [1] 4.286e-09
```

Just like the critical t-value in the t-test, the critical chi-squared value here is:

```
## critical value:
qchisq(0.95,df=1)
```

```
## [1] 3.841
```

If minus two times the observed difference in log likelihood is larger than this critical value, we reject the null hypothesis.

Note that in the likelihood test above, we are comparing one nested model against another: the null hypothesis model is nested inside the alternative hypothesis model. What this means is that the alternative hypothesis model contains all the parameters in the null hypothesis model (i.e., the intercept) plus another one (the slope).

4.3 A real-life example: The English relative clause data

The likelihood ratio test is also the way that hypothesis testing is done with the linear mixed model. Here is how it works. Let's look again at the Grodner and Gibson (2005) English relative clause data. The null hypothesis here refers to the slope parameter. When we have the sum contrast coding, the intercept β_0 refers to the grand mean, and the slope β_1 is the amount by which subject and object relative clause mean reading times deviate from the grand mean. Testing the null hypothesis that β_1 is 0 amounts to testing whether there is any difference in means between the two relative clause types. This becomes clear if we consider the following.

Let object relatives be coded as $+1$ and subject relatives as -1. Then, the mean reading time μ_{or} for object relatives in the linear mixed model is:

$$\mu_{or} = \beta_0 + \beta_1 \tag{4.18}$$

Similarly, the mean reading time μ_{sr} for subject relatives is:

$$\mu_{sr} = \beta_0 - \beta_1 \tag{4.19}$$

If the null hypothesis is that $\mu_{or} - \mu_{sr} = 0$, then this amounts to saying that:

$$(\beta_0 + \beta_1) - (\beta_0 - \beta_1) = 0 \tag{4.20}$$

Removing the brackets gives us:

$$\beta_0 + \beta_1 - \beta_0 + \beta_1 = 0 \tag{4.21}$$

This yields the equation:

$$2\beta_1 = 0 \tag{4.22}$$

Dividing both sides of the equation by 2, we get the null hypothesis that $\beta_1 = 0$.

Incidentally, if we had rescaled the contrast coding to be not ± 1 but $\pm 1/2$, the parameter β_1 would represent exactly the difference between the two means, and null hypothesis in equation (4.22) would have come out to be $\beta_1 = 0$. This is why it is sometimes better to recode the contrasts as $\pm 1/2$ rather than ± 1. See Schad et al. (2020a) for details; we will discuss this in the contrast coding chapter as well.

Let's load the data, set up the contrast coding, and fit the null versus the alternative models. We will fit varying intercept and varying slopes for subject and item, without correlations for items. We don't attempt to fit a model with by-items varying intercepts and slopes with a correlation because we would get a singularity in the variance covariance matrix.

```
gg05e1<-read.table("data/grodnergibsonE1crit.txt",header=TRUE)

gg05e1$so <- ifelse(gg05e1$condition=="objgap",1,-1)
gg05e1$logrt<-log(gg05e1$rawRT)

library(lme4)
m0<-lmer(logrt~1 + (1+so|subject)+(1+so||item),gg05e1)
m1<-lmer(logrt~1 + so + (1+so|subject)+(1+so||item),gg05e1)
```

Notice that we keep all random effects in the null model. We say that the null model is nested inside the full model.

Next, we compare the two models' log likelihoods. There is a function in the `lme4` package that achieves that: the `anova` function:

```
anova(m0,m1)
```

```
## refitting model(s) with ML (instead of REML)
```

```
## Data: gg05e1
## Models:
## m0: logrt ~ 1 + (1 + so | subject) + ((1 | item) + (0 + so | item))
## m1: logrt ~ 1 + so + (1 + so | subject) + ((1 | item) + (0 + so |
## m1:     item))
##    npar AIC BIC logLik deviance Chisq Df Pr(>Chisq)
## m0    7 707 739   -347      693
## m1    8 703 739   -343      687  6.15  1      0.013
```

You can confirm from the output that the `Chisq` value shown is minus two times the difference in log likelihood of the two models. The p-value is computed using the chi-squared distribution with one degree of freedom because in the two models the difference in the number of parameters is one:

```
round(pchisq(6.15,df=1,lower.tail=FALSE),3)
```

```
## [1] 0.013
```

It is common in the psycholinguistics literature to use the t-value from the linear mixed model output to conduct the hypothesis test on the slope:

```
summary(m1)$coefficients
```

```
##             Estimate Std. Error t value
## (Intercept)  5.88306    0.05176 113.669
## so           0.06202    0.02422   2.561
```

The most general method for hypothesis testing is the likelihood ratio test shown above. One can use the t-test output from the linear mixed model for hypothesis testing, but this should be done only when the data are balanced. If there is lack of balance (e.g., missing data for whatever reason), the likelihood ratio test is the best way to proceed. In any case, when we talk about the evidence against the null hypothesis, the likelihood ratio test is the only reasonable way to talk about what evidence we have. See Royall (1997) for more discussion of this point. The essence of Royall's point is that the most reasonable way to talk about the evidence in favor of a particular model is with reference to, i.e., relative to, a baseline model.

One can also use the likelihood ratio test to evaluate whether a variance component should be included or not. For example, is the correlation parameter justified for the subjects random effects? Recall that we had a correlation of 0.58. Is this statistically significant? One can test this in the following way:

```
m1<-lmer(logrt~1 + so + (1+so|subject)+(1+so||item),gg05e1)
m1NoCorr<-lmer(logrt~1 + so + (1+so||subject)+(1+so||item),gg05e1)
anova(m1,m1NoCorr)
```

```
## refitting model(s) with ML (instead of REML)
```

```
## Data: gg05e1
## Models:
## m1NoCorr: logrt ~ 1 + so + ((1 | subject) + (0 + so | subject)) + ((1 |
## m1NoCorr:     item) + (0 + so | item))
## m1: logrt ~ 1 + so + (1 + so | subject) + ((1 | item) + (0 + so |
## m1:     item))
##          npar AIC BIC logLik deviance Chisq Df
## m1NoCorr    7 710 741   -348      696
## m1          8 703 739   -343      687   8.7  1
##          Pr(>Chisq)
## m1NoCorr
## m1           0.0032
```

The test indicates that we can reject the null hypothesis that the correlation parameter is 0. We will return to this parameter in the chapter on simulation.

A final point: the likelihood ratio has essentially the same logic as the t-test, and that includes the fact that the focus is on the rejection of the null. The null cannot be accepted in the face of a null result, unless there is a good case to be made that power is sufficiently high (we will investigate this point later, in the simulation chapter). Also, since the likelihood ratio depends on defining the alternative model using the maximum likelihood estimate (the sample mean), it suffers from exactly the same problem as the t-test (Type M errors in the face of low power). We will also investigate this point in the simulation chapter.

4.4 Exercises

4.4.1 Chinese relative clauses

Load the following two data-sets:

```
gibsonwu<-read.table("data/gibsonwucrit.txt",
                     header=TRUE)
gibsonwu2<-read.table("data/gibsonwu2012datarepeat.txt",
                      header=TRUE)
```

The data are taken from two experiments that investigate (inter alia) the effect of relative clause type on reading time in Chinese. The data are from Gibson and Wu (2013) and Vasishth et al. (2013) respectively. The second data-set is a direct replication attempt of the first.

Chinese relative clauses are interesting theoretically because they are prenominal: the relative clause appears before the head noun.

As iscussed in Gibson and Wu (2013), the consequence of Chinese relative clauses being prenominal is that the distance between the

gap in relative clause and the head noun is larger in subject relatives than object relatives. Hsiao and Gibson (2003) were the first to suggest that the larger distance in subject relatives leads to longer reading time at the head noun. Under this view, the prediction is that subject relatives are harder to process than object relatives. If this is true, this is interesting because in most other languages that have been studied, subject relatives are easier to process than object relatives; so Chinese will be a very unusual exception cross-linguistically.

The data provided are for the critical region (the head noun). The experiment method is self-paced reading, so we have reading times in milliseconds.

The research hypothesis is whether the difference in reading times between object and subject relative clauses is negative. For both data-sets, investigate this question by (a) fitting a paired t-test (by-subjects and by items), (b) fitting the most complex linear mixed model you can to the data and then interpreting the t-value, and (c) the likelihood ratio test. What can we conclude about the research question?

4.4.2 Agreement attraction in comprehension

Load the following data:

```
datE1<-read.table("data/dillonE1.txt",header=TRUE)
```

The data are taken from an experiment that investigate (inter alia) the effect of number similarity between a noun and the auxiliary verb in sentences like the following. There are two levels to a factor called Int(erference): low and high.

(a) low: The key to the cabinet *are* on the table
(b) high: The key to the *cabinets are* on the table

Here, in (b), the auxiliary verb *are* is predicted to be read faster than in (a), because the plural marking on the noun *cabinets*

leads the reader to think that the sentence is grammatical. (Note that both sentences are ungrammatical.) This phenomenon, where the high condition is read faster than the low condition, is called **agreement attraction**.

The data provided are for the critical region (the auxiliary verb *are*). The experiment method is eyetracking; we have total reading times in milliseconds.

The research question is whether the difference in reading times between high and low conditions is negative.

- First, figure out which linear mixed model is appropriate for these data (varying intercepts only? varying intercepts and slopes? with or without correlations?).
- Then, carry out a statistical test using (a) the paired t-test (using the t.test function), (b) the t-test of the linear mixed model, and (c) the likelihood ratio test. What is your conclusion? Is there evidence for agreement attraction in the data?

4.4.3 The grammaticality illusion

Load the following data-sets:

```
english<-read.table("data/embeddingenglish.txt",
                    header=TRUE)
dutch<-read.table("data/embeddingdutch.txt",
                  header=TRUE)
```

In an offline accuracy rating study on English double center-embedding constructions, Gibson and Thomas (1999) found that grammatical constructions (e.g., example a below) were no less acceptable than ungrammatical constructions (e.g., example b) where a middle verb phrase (e.g., was cleaning every week) was missing.

(a) The apartment that the maid who the service had sent over was cleaning every week was well decorated.

(b) *The apartment that the maid who the service had sent over — was well decorated

Based on these results from English, Gibson and Thomas (1999) proposed that working-memory overload leads the comprehender to forget the prediction of the upcoming verb phrase (VP), which reduces working-memory load. This came to be known as the *VP-forgetting hypothesis*. The prediction is that in the word immediately following the final verb, the grammatical condition (which is coded as +1 in the data-frames) should be harder to read than the ungrammatical condition (which is coded as -1).

The data provided above test this hypothesis using self-paced reading for English (Vasishth et al., 2011), and for Dutch (Frank et al., 2015). The data provided are for the critical region (the noun phrase, labeled NP1, following the final verb). We have reading times in log milliseconds.

Is there support for the VP-forgetting hypothesis cross-linguistically, from English and Dutch?

5

Linear modeling theory

So far, we have been studying linear models and linear mixed models rather informally. This approach is good enough for a first-pass at linear modeling. However, in order to fully understand the underlying machinery of linear models, the matrix formulation of the linear model is extremely important and useful. There is a price to be paid, however: one has to reactivate one's knowledge of matrix algebra, or one has to put in some work into reviewing elementary matrix theory. Fortunately, reviewing these concepts is not too difficult. We begin this chapter with a quick review of some basic concepts in matrix algebra, and then apply them to linear modeling. One very useful set of online lectures on the basic ideas behind linear algebra is the short set of videos by 3Blue1Brown: https://youtu.be/kjBOesZCoqc. You should look at these videos before reading the next section, if you have never encountered the basic concepts behind matrix algebra.

5.1 A quick review of some basic concepts in matrix algebra

A rectangular array of numbers (real numbers in our case), with n rows and m columns, is a matrix. The main application for us will be in solving systems of linear equations in statistics.

An example of a 2×2 matrix:

```
## a 2x2 matrix:
(m1<-matrix(1:4,2,2))
```

```
##      [,1] [,2]
## [1,]    1    3
## [2,]    2    4
```

An important operation on matrices is the transpose: the rows become columns and the columns rows. If X is matrix, X^T is its transpose. R has a function `t()` that transposes a matrix:

```
## transpose of a matrix:
t(m1)
```

```
##      [,1] [,2]
## [1,]    1    2
## [2,]    3    4
```

5.1.1 Matrix addition, subtraction, and multiplication

Matrix addition and subtraction are easy, they are cell-wise operations. An example shows how this works:

```
m1+m1
```

```
##      [,1] [,2]
## [1,]    2    6
## [2,]    4    8
```

```
m1-m1
```

```
##      [,1] [,2]
## [1,]    0    0
## [2,]    0    0
```

Matrix multiplication is defined as follows. Think of this simple situation first, where you have a vector of values:

```
(m2<-rnorm(5))
```

```
## [1] -0.17650 -0.99922 -1.24649 -0.75233 -0.07495
```

```
## this is a vector:
str(m2)
```

```
##  num [1:5] -0.177 -0.999 -1.246 -0.752 -0.075
```

If you want to find out the sum of the squares of these values ($\sum_{i=1}^{n} x_i^2$), then you can multiply each value x_i in m2 with itself, and sum up the result. This is called a **dot product** of two vectors (here, $m2$ repeated twice):

$x_1 \cdot x_1 + x_2 \cdot x_2 + x_3 \cdot x_3 + x_4 \cdot x_4 + x_5 \cdot x_5$

Using R:

```
(sum(m2*m2))
```

```
## [1] 3.155
```

An important observation we will need later is that if two vectors are orthogonal, i.e., at 90 degrees to one another, then their dot product is 0. A simple example is two vectors on the cartesian plane that go along the x and y axes:

```
v1<-c(0,1)
v2<-c(1,0)
sum(v1*v2)
```

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

Matrix multiplication does exactly the above dot product operation (for arbitrarily large matrices). In R, matrix multiplication is carried out using a special operator (the * operator we are used to using for multiplication is used only for ordinary scalar multiplication). Notice that we have first converted the vector with 5 cells into a 5×1 matrix:

```
t(matrix(m2))
```

```
##          [,1]    [,2]   [,3]    [,4]     [,5]
## [1,] -0.1765 -0.9992 -1.246 -0.7523 -0.07495
```

```
matrix(m2)
```

```
##           [,1]
## [1,] -0.17650
## [2,] -0.99922
## [3,] -1.24649
## [4,] -0.75233
## [5,] -0.07495
```

```
t(matrix(m2))%*%matrix(m2)
```

```
##       [,1]
## [1,] 3.155
```

Note that two matrices can only be multiplied if they are **conformable**: the number of columns of the first matrix has to be the same as the number of rows of the second matrix. So, if you have an 1×5 matrix, you can multiply it with an 5×1 matrix, and the end result of the multiplication will be an 1×1 matrix (i.e., a scalar value). In the above example, the vector m2, converted to a matrix, is 5×1 in dimensions; you cannot multiply a 5×1 with a 5×1 matrix, but the transform of m2 to a 1×5 matrix can be multiplied with the 5×1 matrix.

More generally, if you have an $n \times m$ matrix and multiply it with an $m \times p$ matrix, you will get a $n \times p$ matrix, and the operations we have to do is the dot product operation, repeatedly applied to each row of the first matrix and each column of the second matrix. For example:

```
X1<-matrix(rnorm(4),ncol=2)
X2<-matrix(rnorm(4),ncol=2)
## matrix multiplication:
X1%*%X2
```

```
##        [,1]   [,2]
## [1,] 1.680 -1.677
## [2,] 2.655 -2.610
```

```
## is a series of dot products of vectors:
sum(X1[1,] * X2[,1])
```

```
## [1] 1.68
```

```
sum(X1[1,] * X2[,2])
```

```
## [1] -1.677
```

```
sum(X1[2,] * X2[,1])
```

```
## [1] 2.655
```

```
sum(X1[2,] * X2[,2])
```

```
## [1] -2.61
```

Scalar matrix multiplication is easy. Multiplying a matrix M with a scalar value just amounts to multiplying the scalar with each cell of the matrix. The order of multiplication doesn't matter:

```
5*m2
```

```
## [1] -0.8825 -4.9961 -6.2324 -3.7617 -0.3748
```

```
m2*5
```

```
## [1] -0.8825 -4.9961 -6.2324 -3.7617 -0.3748
```

5.1.2 Diagonal matrix and identity matrix

A diagonal matrix is a square matrix that has zeros in its off-diagonals:

```
diag(c(1,2,4))
```

```
##      [,1] [,2] [,3]
## [1,]    1    0    0
## [2,]    0    2    0
## [3,]    0    0    4
```

An identity matrix is a diagonal matrix with 1's along its diagonal:

```
diag(rep(1,3))
```

```
##      [,1] [,2] [,3]
## [1,]    1    0    0
## [2,]    0    1    0
## [3,]    0    0    1
```

For a 3×3 identity matrix, we can write I_3, and for an nxn identity matrix, I_n.

Multiplying an identity matrix with any (conformable) matrix gives that matrix back:

```
(I<-diag(c(1,1)))
```

```
##      [,1] [,2]
## [1,]    1    0
## [2,]    0    1
```

```
(m3<-matrix(c(6,7,8,9),2,2))
```

```
##      [,1] [,2]
## [1,]    6    8
## [2,]    7    9
```

```
(I%*%m3)
```

```
##      [,1] [,2]
## [1,]    6    8
## [2,]    7    9
```

```
## the matrix does not have to be square:
(m4<-matrix(c(2,3,6,7,8,9),2,3))
```

```
##      [,1] [,2] [,3]
## [1,]    2    6    8
## [2,]    3    7    9
```

```
(I%*%m4)
```

```
##      [,1] [,2] [,3]
## [1,]    2    6    8
## [2,]    3    7    9
```

5.1.3 Powers of matrices

If A is a square $n \times n$ matrix, then we write AA as A^2, and so on. If A is diagonal, then AA is just the diagonal matrix with the diagonal elements of A squared:

```
m<-diag(c(1,2,3))
m%*%m
```

```
##      [,1] [,2] [,3]
```

```
## [1,]    1    0    0
## [2,]    0    4    0
## [3,]    0    0    9
```

For all positive integers m, $I_n^m = I_n$.

5.1.4 Inverse of a matrix

If A, B are square matrices, of order $n \times n$, and the following relation is satisfied:

$$AB = BA = I_n \tag{5.1}$$

then, B is the inverse (it is unique) of A. We write $B = A^{-1}$. The inverse is analogous to division and allows us to solve matrix equations: AB=C can be solved by post-multiplying both sides by B^{-1} to get $ABB^{-1} = AI = A = CB^{-1}$.

Pre-multiplying a matrix like A above by its inverse A^{-1} gives an identity matrix (the R function `solve` computes the inverse of a matrix):

```
(m5<-matrix(c(2,3,4,5),2,2))
```

```
##      [,1] [,2]
## [1,]    2    4
## [2,]    3    5
```

```
(round(solve(m5)%*%m5))
```

```
##      [,1] [,2]
## [1,]    1    0
## [2,]    0    1
```

If a matrix is not invertible, we say that it is **singular**.

A useful fact to know about non-singular matrices is the following relationship. If A and B are non-singular matrices then $(AB)^{-1} = B^{-1}A^{-1}$.

5.1.5 Linear independence, and rank

An important concept in matrices is linear independence. Consider a 3×3 matrix. The rows r_1, r_2, r_3 are linearly **dependent** if α, β, γ, not all zero, exist such that $\alpha r_1 + \beta r_2 + \gamma r_3 = (0, 0, 0)$.

If the rows or columns of a matrix A are linearly dependent, then the matrix is singular, i.e., it is not invertible.

Linear independence leads us to the concept of the rank of a matrix. The column rank of a matrix is the maximum number of linearly independent columns in the matrix. The row rank is the maximum number of linearly independent rows. Column rank is always equal to row rank, so we can just call it rank.

The above is the minimum vocabulary we need to understand the linear model in its matrix form. Let us now turn to the matrix formulation of the linear model.

5.2 The essentials of linear modeling theory

Consider a deterministic function $\phi(\mathbf{x}, \beta)$ which takes as input some variable values x and some fixed values β. A simple example would be some variable x determining the value of another variable y by multiplying x with β.

$$y = \beta x \tag{5.2}$$

Another example with two fixed values β_0 and β_1 determining y is:

$$y = \beta_0 + \beta_1 x \tag{5.3}$$

We can rewrite the above equation in matrix form as follows.

$$\begin{aligned} y =& \beta_0 + \beta_1 x \\ =& \beta_0 \times 1 + \beta_1 x \\ =& \begin{pmatrix} 1 & x \end{pmatrix} \begin{pmatrix} \beta_0 \\ \beta_1 \end{pmatrix} \end{aligned} \tag{5.4}$$

Because y is a function of x and the 2×1 matrix $\beta = [\beta_0 \beta_1]^T$, the most general way to write the above function is as we did above:

$$y = \phi(x, \beta) \tag{5.5}$$

In a statistical model, we don't expect an equation like $y = \phi(x, \beta)$ to fit all the points exactly. For example, we could come up with an equation that, given a word's frequency, gives a prediction regarding that word's reaction time:

$$\text{predicted reaction time} = \beta_0 + \beta_1 \text{frequency} \tag{5.6}$$

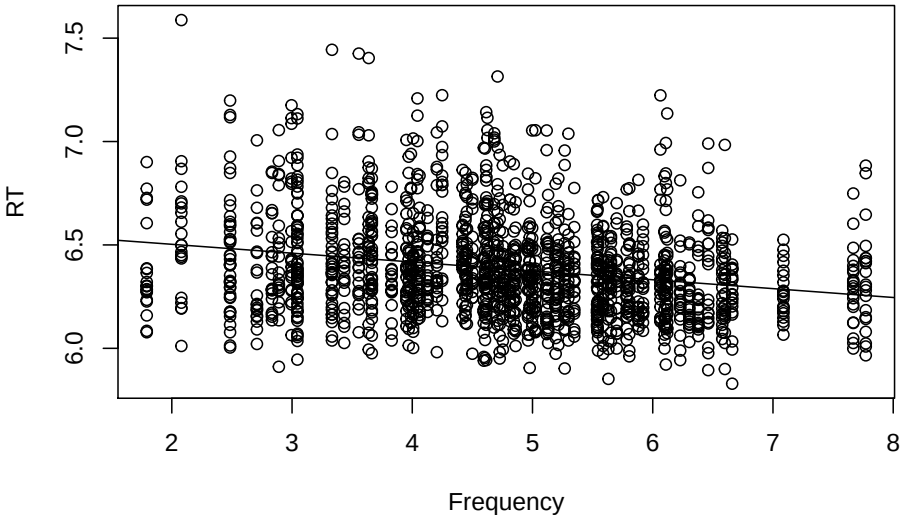
Given any single value of the frequency of a word, we will not get a perfectly correct prediction of the reaction time for that word. As a concrete example, see this data-set from the `languageR` library, which allows us to visualize the effect of (log) word frequency on (log) reaction times. This model is technically incorrect, because we have repeated measures; a linear mixed model would be more appropriate. But a simple linear model is sufficient to make the point here:

```
library(languageR)
```

```
data("lexdec")
m<-lm(RT~Frequency,lexdec)
summary(m)$coefficients
```

```
##             Estimate Std. Error t value  Pr(>|t|)
## (Intercept)  6.58878   0.022296 295.515 0.000e+00
## Frequency   -0.04287   0.004533  -9.459 1.027e-20
```

```
plot(RT~Frequency,lexdec)
abline(m)
```



Because the predicted values from the linear model don't exactly predict the observed vlues, we express the dependent variable y as a non-deterministic function:

$$y = \phi(x, \beta, \epsilon) = \beta_0 + \beta_1 x + \epsilon \tag{5.7}$$

Here, ϵ is an error random variable which is assumed to have some PDF (the normal distribution) associated with it. It is assumed to have expectation (mean) 0, and some standard deviation (to be estimated from the data) σ. We can write this statement in compact form as $\epsilon \sim N(0, \sigma)$.

The **general linear model** is a non-deterministic function like the one above:

$$Y = f(x)^T \beta + \epsilon \tag{5.8}$$

The matrix formulation will be written as below. n refers to the number of data points (that is, $Y_1, \ldots, Y_n$), and the index j ranges from 1 to n.

$$Y = X\beta + \epsilon \Leftrightarrow y_j = f(x_j)^T\beta + \epsilon_j, j = 1, \ldots, n \tag{5.9}$$

To make this concrete, suppose we have three data points, i.e., $n = 3$. Then, the matrix formulation is

$$\begin{pmatrix} Y_1 \\ Y_2 \\ Y_3 \end{pmatrix} = \begin{pmatrix} 1 & x_1 \\ 1 & x_2 \\ 1 & x_3 \end{pmatrix} \begin{pmatrix} \beta_0 \\ \beta_1 \end{pmatrix} + \begin{pmatrix} \epsilon_1 \\ \epsilon_2 \\ \epsilon_3 \end{pmatrix} \tag{5.10}$$

We can write this in compact form as follows:

$$Y = X\beta + \epsilon \tag{5.11}$$

Y is a 3×1 matrix, X is a 3×2 matrix, β 2×1, and ϵ 3×1.

Here, $f(x_1)^T = (1\ x_1)$, and is the first row of the matrix X, $f(x_2)^T = (1\ x_2)$ is the second row, and $f(x_3)^T = (1\ x_3)$ is the third row.

Note that the expectation or mean of Y, written $E[Y]$, is $X\beta$. In a data-set with n data points, when there are p parameters, β is a $p \times 1$ matrix, and X, which is called the **design matrix** or **model matrix**, is $n \times p$.

5.2.1 Least squares estimation: Geometric argument

The above excursion into the matrix formulation of the linear model gives us the ability to understand how the β parameters are estimated.

When we have a deterministic model $y = \phi(f(x)^T, \beta) = \beta_0 + \beta_1 x$, this implies a perfect fit to all data points. This is like solving the equation $Ax = b$ in linear algebra: we solve for β in $X\beta = y$ e.g.,

by pre-multiplying by X^{-1}: $X^{-1}X\beta = X^{-1}y$. For example, if we have:

```
(X<-matrix(c(1,2,2,2,2,4,6,8,3,6,8,10),nrow =3,byrow=TRUE))
```

```
##      [,1] [,2] [,3] [,4]
## [1,]    1    2    2    2
## [2,]    2    4    6    8
## [3,]    3    6    8   10
```

```
(y<-matrix(c(1,5,6)))
```

```
##      [,1]
## [1,]    1
## [2,]    5
## [3,]    6
```

We can solve for x as follows (ginv is a function that computed a so-called generalized inverse for non-square matrices):

```
library(MASS)
## Solve for x in Ax=y:
x<-ginv(X)%*%y
## confirm:
X%*%x
```

```
##      [,1]
## [1,]    1
## [2,]    5
## [3,]    6
```

But when we have a non-deterministic model $y = \phi(f(x)^T, \beta, \epsilon)$, there is no solution! Now, the best we can do in an equation like $Ax = b$ is to get Ax to be as close an approximation as possible to b. In other words, we try to minimize $\mid b - Ax \mid$.

The goal is to estimate β; we want to find a value of β such that

the observed Y is as close to its expected value $X\beta$. In order to be able to identify β from $X\beta$, the linear transformation $\beta \rightarrow X\beta$ should be one-to-one, so that every possible value of β gives a different $X\beta$. This in turn requires that X be of full rank p. So, if a design matrix X is $n \times p$, then it is necessary that $n \geq p$. There must be at least as many observations as parameters. If this is not true, then the model is said to be **over-parameterized**.

Assuming that X is of full rank, and that $n > p$, Y can be considered a point in n-dimensional space and the set of candidate $X\beta$ is a p-dimensional subspace of this space; see Figure 5.1. There will be one point in this subspace which is closest to Y in terms of Euclidean distance. The unique β that corresponds to this point is the **least squares estimator** of β; we will call this estimator $\hat{\beta}$.

Notice that $\epsilon = (Y - X\hat{\beta})$ and $X\beta$ are perpendicular to each other. Because the dot product of two perpendicular (orthogonal) vectors is 0, we get the result:

$$(Y - X\hat{\beta})^T X\beta = 0 \Leftrightarrow (Y - X\hat{\beta})^T X = 0 \tag{5.12}$$

Multiplying out the terms, we proceed as follows. One result that we use here is that $(AB)^T = B^T A^T$.

$$\begin{aligned} &(Y - X\hat{\beta})^T X = 0 \\ &(Y^T - \hat{\beta}^T X^T)X = 0 \\ \Leftrightarrow &Y^T X - \hat{\beta}^T X^T X = 0 \\ \Leftrightarrow &Y^T X = \hat{\beta}^T X^T X \\ \Leftrightarrow &(Y^T X)^T = (\hat{\beta}^T X^T X)^T \\ \Leftrightarrow &X^T Y = X^T X\hat{\beta} \end{aligned} \tag{5.13}$$

This gives us the important result:

$$\hat{\beta} = (X^T X)^{-1} X^T Y \tag{5.14}$$

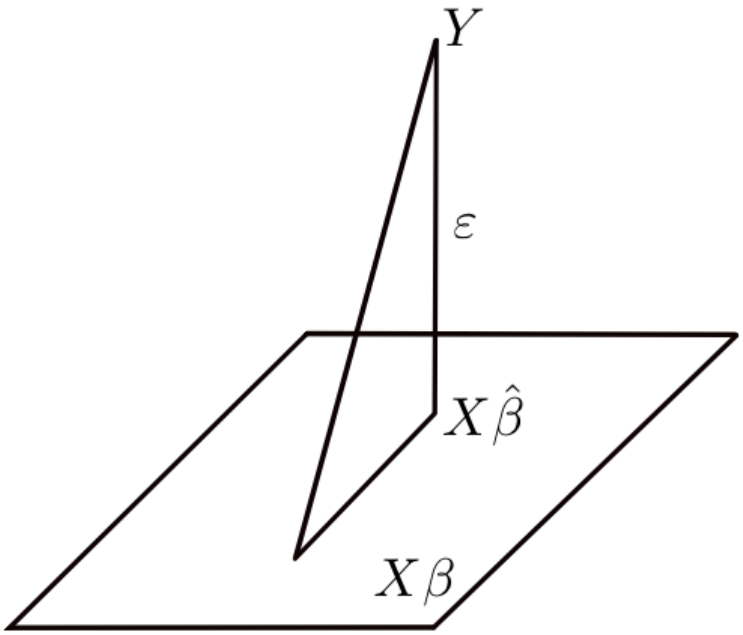


FIGURE 5.1: Geometric visualization of the distance minimization procedure for estimating the parameters in a linear model.

X is of full rank, therefore X^TX is invertible. One crucial detail, whose significance will only become clear in the chapter on contrast coding, is that the matrix $(X^TX)^{-1}X^T$—the generalized matrix inverse—is closely related to the design matrix, which in turn determines what comparisons are implied by the β parameters.

Let's look at a concrete example:

```
(X<-matrix(c(rep(1,8),rep(c(-1,1),each=4),
             rep(c(-1,1),each=2,2)),ncol=3))
```

```
##      [,1] [,2] [,3]
## [1,]    1   -1   -1
## [2,]    1   -1   -1
## [3,]    1   -1    1
## [4,]    1   -1    1
## [5,]    1    1   -1
## [6,]    1    1   -1
## [7,]    1    1    1
## [8,]    1    1    1
```

```
library(Matrix)
## full rank:
rankMatrix(X)
```

```
## [1] 3
## attr(,"method")
## [1] "tolNorm2"
## attr(,"useGrad")
## [1] FALSE
## attr(,"tol")
## [1] 1.776e-15
```

Notice that the inverted matrix is also symmetric. We will use this fact soon.

The matrix $V = X^TX$ is a symmetric matrix, which means that $V^T = V$.

5.2.2 The expectation and variance of the parameters beta

Our model now is:

$$Y = X\beta + \epsilon \tag{5.15}$$

Let $\epsilon \sim N(0, \sigma)$. In other words, we are assuming that each value generated by the random variable ϵ is independent and it has the same distribution, i.e., it is identically distributed. This is sometimes shortened to the iid assumption. So we should technically be writing:

$$Y = X\beta + \epsilon \quad \epsilon \sim N(0, \sigma) \tag{5.16}$$

and add that Y are independent and identically distributed.

Some consequences of the above statements:

- $E[\epsilon] = 0$
- $Var(\epsilon) = \sigma^2 I_n$
- $E[Y] = X\beta = \mu$
- $Var(Y) = \sigma^2 I_n$

We can now derive the expectation and variance of the estimators $\hat{\beta}$. We need a fact about variances: when we want to know $Var(a \times B)$, where a is a constant and B is a random variable, this variance is $a^2 Var(B)$. In the matrix setting, Var(AB), where A is a conformable matrix consisting of some constant values, is $AVar(B)A^T$.

$$E[\hat{\beta}] = E[(X^T X)^{-1} X^T Y] = (X^T X)^{-1} X^T X\beta = \beta \tag{5.17}$$

Notice that the above shows that $\hat{\beta}$ is a so-called "unbiased estimator" of β. The word unbiased doesn't mean that every time you compute an estimate of β, you are guaranteed to get an accurate estimate of the true β! Think about Type M error.

Next, we compute the variance:

$$Var(\hat{\beta}) = Var([(X^T X)^{-1} X^T]Y) \tag{5.18}$$

Expanding the right hand side out:

$$Var([(X^T X)^{-1} X^T]Y) = [(X^T X)^{-1} X^T]Var(Y)[(X^T X)^{-1} X^T]^T \tag{5.19}$$

Replacing Var(Y) with its variance written in matrix form $\sigma^2 I$, and unpacking the transpose on the right-most expression $[(X^T X)^{-1} X^T]^T$:

$$Var(\hat{\beta}) = [(X^T X)^{-1} X^T]\sigma^2 I X[(X^T X)^{-1}]^T \tag{5.20}$$

Since σ^2 is a scalar we can move it to the left, and any matrix multiplied by I is the matrix itself, so we ignore I, getting:

$$Var(\hat{\beta}) = \sigma^2[(X^T X)^{-1} X^T X[(X^T X)^{-1}]^T \tag{5.21}$$

Since $(X^T X)^{-1} X^T X = I$, we can simplify to

$$Var(\hat{\beta}) = \sigma^2[(X^T X)^{-1}]^T \tag{5.22}$$

Now, $(X^T X)^{-1}$ is symmetric, so $[(X^T X)^{-1}]^T = (X^T X)^{-1}$. This gives us:

$$Var(\hat{\beta}) = \sigma^2(X^T X)^{-1} \tag{5.23}$$

Let's make this concrete using the `lexdec` data-set as an example:

```
y<-lexdec$RT
x<-lexdec$Frequency
m0<-lm(y~x)
```

```
## design matrix:
X<-model.matrix(m0)
head(X,n=4)
```

```
##   (Intercept)     x
## 1           1 4.860
## 2           1 4.605
## 3           1 4.997
## 4           1 4.727
```

```
## (X^TX)^{-1}
invXTX<-solve(t(X)%*%X)
## estimate of beta:
(hat_beta<-invXTX%*%t(X)%*%y)
```

```
##                 [,1]
## (Intercept)  6.58878
## x           -0.04287
```

```
## estimated variance (se^2) of the estimate of beta:
(hat_sigma<-summary(m0)$sigma)
```

```
## [1] 0.2353
```

```
(hat_var<-hat_sigma^2*invXTX)
```

```
##             (Intercept)          x
## (Intercept)   0.0004971 -9.760e-05
## x            -0.0000976  2.054e-05
```

What we have here is a bivariate normal distribution as an estimate of the β parameters:

$$\begin{pmatrix}\hat{\beta}_0\\ \hat{\beta}_1\end{pmatrix} \sim N(\begin{pmatrix}6.58878\\ -0.04287\end{pmatrix}, \begin{pmatrix}0.000497 & -0.0000976\\ -0.0000976 & 2.054e-05\end{pmatrix}) \quad (5.24)$$

The variance of a bivariate distribution has the variances along the diagonal, and the covariance between $\hat{\beta}_0$ and $\hat{\beta}_1$ on the off-diagonals. Covariance is defined as:

$$Cov(\hat{\beta}_0, \hat{\beta}_1) = \hat{\rho}\hat{\sigma}_{\hat{\beta}_0}\hat{\sigma}_{\hat{\beta}_1} \quad (5.25)$$

where $\hat{\rho}$ is the estimated correlation between β_0 and β_1.

So

- $\hat{\beta}_0 \sim N(6.588778, 0.022296)$
- $\hat{\beta}_1 \sim N(-0.042872, 0.0045325)$, and - $Cov(\hat{\beta}_0, \hat{\beta}_1) = -9.76 \times 10^{-05}$.

So the correlation between the $\hat{\beta}$ is

```
## hat rho:
hat_var[1,2]/(sqrt(hat_var[1,1])*sqrt(hat_var[2,2]))
```

```
## [1] -0.9658
```

Notice what happens to this correlation when we center the predictor:

```
x_c<-scale(x,scale=FALSE)
m<-lm(y~x_c)
## design matrix:
X<-model.matrix(m)
## (X^TX)^{-1}
invXTX<-solve(t(X)%*%X)
## estimate of beta:
(hat_beta<-invXTX%*%t(X)%*%y)
```

```
##                  [,1]
## (Intercept)  6.38509
## x_c         -0.04287
```

```
## estimated variance (se^2) of the estimate of beta:
(hat_sigma<-summary(m0)$sigma)
```

```
## [1] 0.2353
```

```
(hat_var<-hat_sigma^2*invXTX)
```

```
##             (Intercept)       x_c
## (Intercept)   3.338e-05 7.204e-21
## x_c           7.204e-21 2.054e-05
```

```
## correlation:
round(hat_var[1,2]/(sqrt(hat_var[1,1])*sqrt(hat_var[2,2])),4)
```

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

The correlation now is (effectively) zero. This is one of the consequences of centering your predictor: the intercept-slope sampling distributions become independent. The relevance of this fact will be discussed in the chapters on contrast coding.

5.2.3 Hypothesis testing using Analysis of variance (ANOVA)

We can compare two models, one nested inside another, as follows:

```
m1<-lm(y~x_c)
m0<-lm(y~1)
anova(m0,m1)
```

```
## Analysis of Variance Table
##
## Model 1: y ~ 1
```

```
## Model 2: y ~ x_c
##   Res.Df  RSS Df Sum of Sq    F Pr(>F)
## 1   1658 96.7
## 2   1657 91.8  1      4.95 89.5 <2e-16
```

The F-score you get here is actually the square of the t-value you get in the linear model summary:

```
sqrt(anova(m0,m1)$F[2])
```

```
## [1] 9.459
```

```
summary(m1)$coefficients[2,3]
```

```
## [1] -9.459
```

This is because $t^2 = F$. The proof is discussed on page 9 of the Dobson and Barnett book, but we also briefly alluded to this in the likelihood ratio test chapter.

The ANOVA works as follows. We will use the matrix formulation throughout. First define the residual as:

$$e = Y - X\hat{\beta} \tag{5.26}$$

The square of this is:

$$e^T e = (Y - X\hat{\beta})^T (Y - X\hat{\beta}) \tag{5.27}$$

Define the **deviance** as:

$$\begin{aligned} D =& \frac{1}{\sigma^2}(Y - X\hat{\beta})^T (Y - X\hat{\beta}) \\ =& \frac{1}{\sigma^2}(Y^T - \hat{\beta}^T X^T)(Y - X\hat{\beta}) \\ =& \frac{1}{\sigma^2}(Y^T Y - Y^T X\hat{\beta} - \hat{\beta}^T X^T Y + \hat{\beta}^T X^T X\hat{\beta}) \end{aligned} \tag{5.28}$$

Now, recall that $\hat{\beta} = (X^T X)^{-1} X^T Y$. Premultiplying both sides with $(X^T X)$, we get

$(X^T X)\hat{\beta} = X^T Y$

It follows that we can rewrite the last line in equation 5.28 as follows: We can replace $(X^T X)\hat{\beta}$ with $X^T Y$.

$$\begin{aligned} D =& \frac{1}{\sigma^2}(Y^T Y - Y^T X\hat{\beta} - \hat{\beta}^T X^T Y + \hat{\beta}^T \underline{X^T X\hat{\beta}}) \\ =& \frac{1}{\sigma^2}(Y^T Y - Y^T X\hat{\beta} - \hat{\beta}^T X^T Y + \hat{\beta}^T \underline{X^T Y}) \\ =& \frac{1}{\sigma^2}(Y^T Y - Y^T X\hat{\beta}) \end{aligned} \tag{5.29}$$

Notice that $Y^T X\hat{\beta}$ is a scalar (1×1) and is identical to $\beta^T X^T Y$ (check this), so we could write:

$D = \frac{1}{\sigma^2}(Y^T Y - \hat{\beta}^T X^T Y)$

Assume now that we have data of size n. Suppose we have a null hypothesis $H_0 : \beta = \beta_0$ and an alternative hypothesis $H_1 : \beta = \beta_1$. Let the null hypothesis have q parameters, and the alternative p, where $q < p < n$. Let X_0 be the design matrix for H_0, and X_1 the design matrix for H_1. Compute the deviances D_0 and D_1 for each hypothesis, and compute ΔD:

$$\begin{aligned} \Delta D =& D_0 - D_1 = \frac{1}{\sigma^2}[(Y^T Y - \hat{\beta}_0 X_0^T Y) - (Y^T Y - \hat{\beta}_1 X_1^T Y)] \\ =& \frac{1}{\sigma^2}[\hat{\beta}_1 X_1^T Y - \hat{\beta}_0 X_0^T Y] \end{aligned} \tag{5.30}$$

It turns out that the F-statistic has the following distribution (called the F-distribution, defined in terms of two numerical parameters) if the null hypothesis is true:

$$F = \frac{\Delta D/(p-q)}{D_1/(n-p)} \sim F(p-q, n-p) \tag{5.31}$$

So, an observed F value that lies in the tail of the F-distribution is inconsistent with the null hypothesis and allows us to reject it.

The observed F-statistic is:

$$\begin{aligned} F =& \frac{\Delta D/(p-q)}{D_1/(n-p)} \\ =& \frac{\hat{\beta}_1 X_1^T Y - \hat{\beta}_0^T X_0^T Y}{p-q} / \frac{Y^T Y - \hat{\beta}_1^T X_1^T Y}{n-p} \end{aligned} \tag{5.32}$$

Traditionally, the way the F-test is summarized is shown in Table 5.1.

TABLE 5.1: The standard way to summarize an ANOVA analysis.

Source of variance	df	Sum of squares	Mean square
Model with β_0	q	$\hat{\beta}_0^T X_0^T Y$	
Improvement due to β_1	p-q	$\hat{\beta}_1 X_1^T Y - \hat{\beta}_0^T X_0^T Y$	$\frac{\hat{\beta}_1 X_1^T Y - \hat{\beta}_0^T X_0^T Y}{p-q}$
Residual	n-p	$Y^T Y - \hat{\beta}_1^T X_1^T Y$	$\frac{Y^T Y - \hat{\beta}_1^T X_1^T Y}{n-p}$
Total	n	$Y^T Y$	

5.2.4 Some further important topics in linear modeling

5.2.4.1 The variance inflation factor: Checking for multicollinearity

The linear modeling framework is very flexible, and allows us to add multiple predictors at the same time. For example, in the lexical decision data set, we could look at the effect of Frequency and FamilySize:

```
m<-lm(RT~Frequency+FamilySize,lexdec)
summary(m)
```

```
##
## Call:
## lm(formula = RT ~ Frequency + FamilySize, data = lexdec)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -0.5510 -0.1608 -0.0344  0.1204  1.0969
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  6.56385    0.02683  244.68  < 2e-16
## Frequency   -0.03531    0.00641   -5.51  4.1e-08
## FamilySize  -0.01565    0.00938   -1.67    0.095
##
## Residual standard error: 0.235 on 1656 degrees of freedom
## Multiple R-squared:  0.0528, Adjusted R-squared:  0.0517
## F-statistic: 46.2 on 2 and 1656 DF,  p-value: <2e-16
```

An important issue here is **multicollinearity**. This occurs when multiple predictors are highly correlated. The consequence of this is that X^TX can be nearly singular and the estimation equation

$$X^TX\beta = X^TY \tag{5.33}$$

is ill-conditioned: small changes in the data can cause large changes in β (signs will flip for example). Also, some of the elements of $\sigma^2(X^TX)^{-1}$ will be large–standard errors and covariances can be large.

We can check for multicollinearity using the Variance Inflation Factor, VIF.

Suppose you have fitted a model with several predictors. The definition of VIF_j for a predictor j is:

$$VIF_j = \frac{1}{1 - R_j^2} \tag{5.34}$$

where R_j^2 is called the coefficient of determination for predictor j. It is obtained by regressing the j-th explanatory variable against all the other explanatory variables. If a predictor j is uncorrelated with all other predictors, then $VIF_j = 1$, and if it is highly correlated with (some of) other predictors, the VIF will be high.

Here is a practical example. Consider frequency and family size in the lexdec data:

```
library(car)
m<-lm(RT~Frequency+FamilySize,lexdec)
vif(m)
```

```
##  Frequency FamilySize
##          2          2
```

Here is a somewhat worse situation:

```
m<-lm(RT~FreqSingular+FreqPlural,lexdec)
vif(m)
```

```
## FreqSingular   FreqPlural
##        4.681        4.681
```

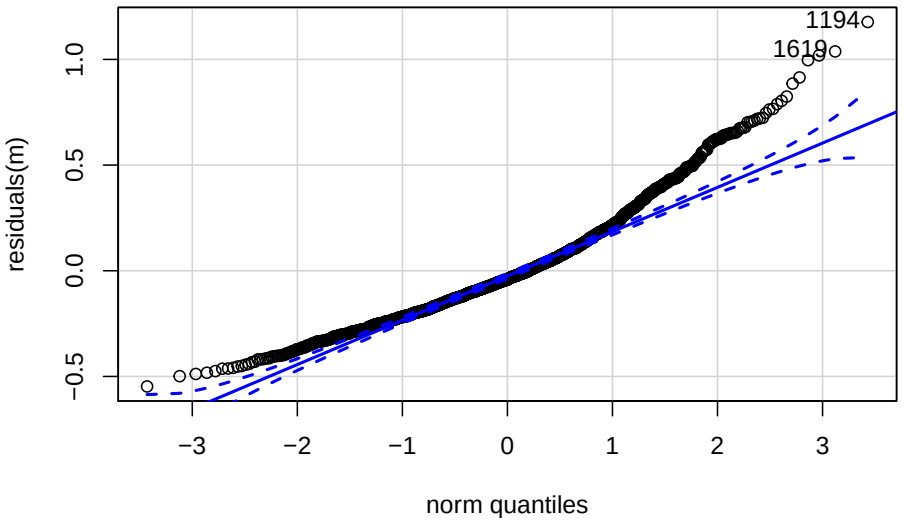
If the predictors are uncorrelated, VIF will be near 1 in each case. Dobson et al mention that VIF of greater than 5 is cause for worry.

5.2.4.2 Checking model assumptions

In practical terms, when doing hypothesis testing with the linear model, the first thing you need to check is whether the residuals are normally distributed. This is because the normality assumption is a crucial part of the assumptions in the hypothesis test (if the goal is only estimation of the parameters, the normality assumption is less important; however, there are subtleties even in this case,

which we will discuss in the simulation chapter). One common way to check for (approximate) normality is to plot the residuals against the quantiles of the normal distribution:

```
library(car)
qqPlot(residuals(m))
```

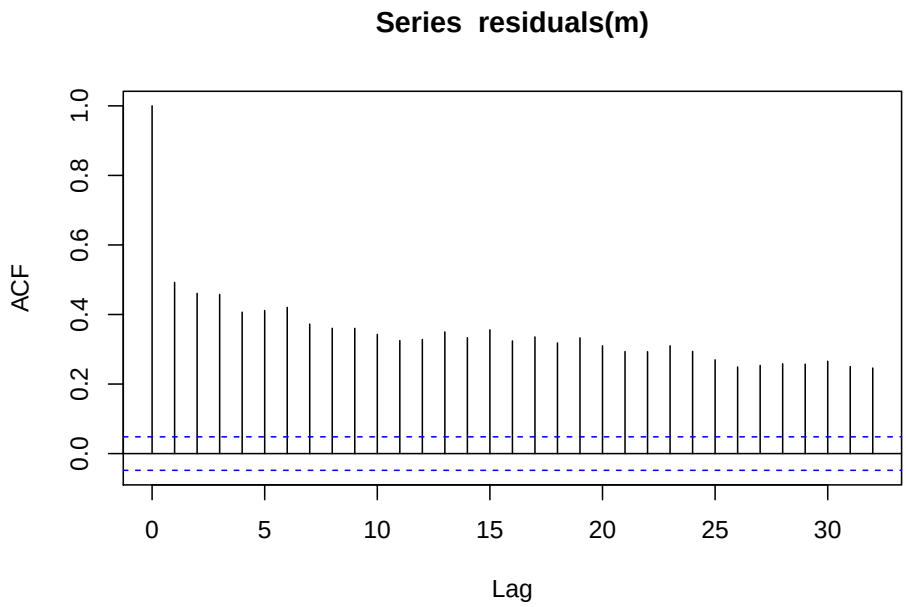



```
## 1194 1619
## 1273  750
```

How to test for normality of residuals? Kolmogorov-Smirnov and Shapiro-Wilk are formal tests of normality and are only useful for large samples; they not very powerful and not much better than diagnostic plots like the qq-plot shown above.

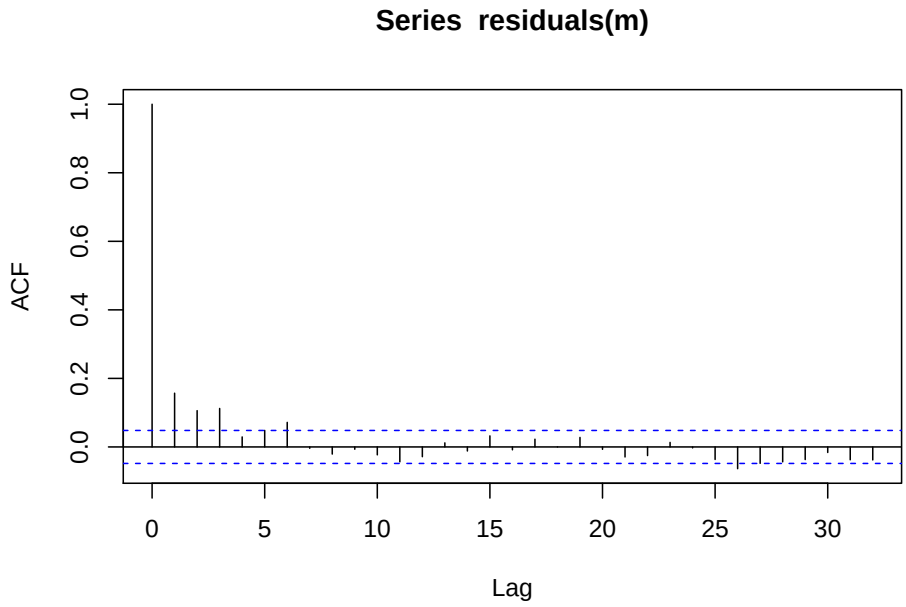
Apart from normality, we should also check the independence assumption (the errors are assumed to be independent). Index-plots display residuals against observation number; note that they are not useful for small samples. An alternative is to compute the correlation between e_i, e_{i+1} pairs of residuals. The auto-correlation function is not normally used in linear modeling (it's used more in time-series analyses), but can be used to check for this correlation:

```
acf(residuals(m))
```



In our model (which is the multiple regression we did in connection with the collinearity issue), we have a serious violation of independence. This is because in this model we are not taking into account the fact that we have repeated measures. The repeated measures create a dependency in the residuals, violating the independence assumption. This problem can be largely solved by fitting a linear mixed model:

```
library(lme4)
m<-lmer(RT~FreqSingular+FreqPlural+(1|Subject),lexdec)
acf(residuals(m))
```

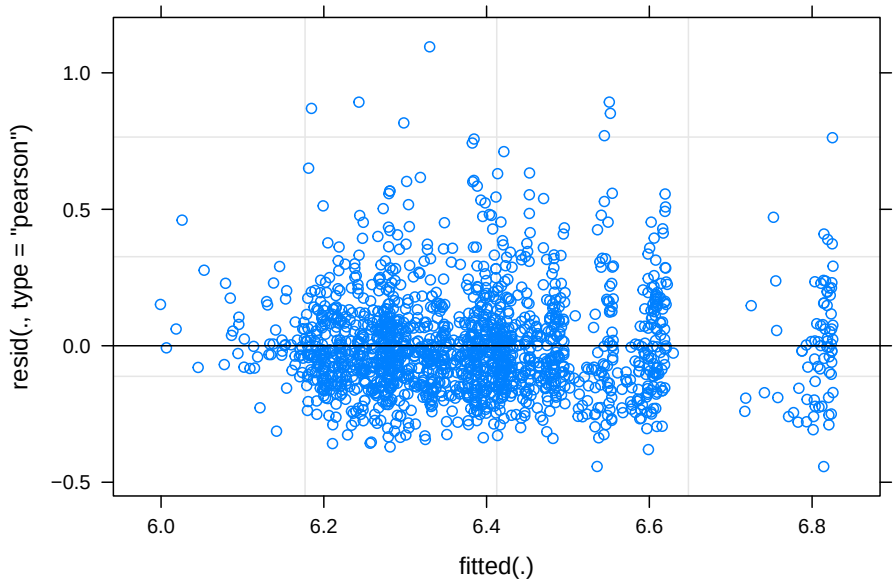


The linear mixed model is taking the repeated measurements property into account.

Finally, we should check for homoscedasticity (equality of variance). For checking this, plot residuals against fitted values. A fanning-out suggests violation. A quadratic trend in a plot of residuals against predictor x could suggest that a quadratic predictor term is needed; note that we will never have a perfect straight line in such a plot.

R also provides a diagnostics plot, which is generated using the model fit:

```
op<-par(mfrow=c(2,2),pty="s")
plot(m)
```



In linear mixed models, there is a package called `influence.ME` (Nieuwenhuis et al., 2012) that allows one to check model assumptions.

5.2.5 Generalized linear models

5.2.5.1 Introduction: Logistic regression

We start with an example data-set that appears in the Dobson et al book: the Beetle data-set. Although not a linguistic or psychological data-set, it is a good entry point into generalized linear models. Later we will apply the theory to psycholinguistic data.

The Beetle data-set shows the number of beetles killed when they were exposed to different doses of some toxic chemical.

```
(beetle<-read.table("data/beetle.txt",header=TRUE))
```

```
##    dose number killed
## 1 1.691     59      6
## 2 1.724     60     13
## 3 1.755     62     18
## 4 1.784     56     28
```

```
## 5 1.811     63     52
## 6 1.837     59     53
## 7 1.861     62     61
## 8 1.884     60     60
```

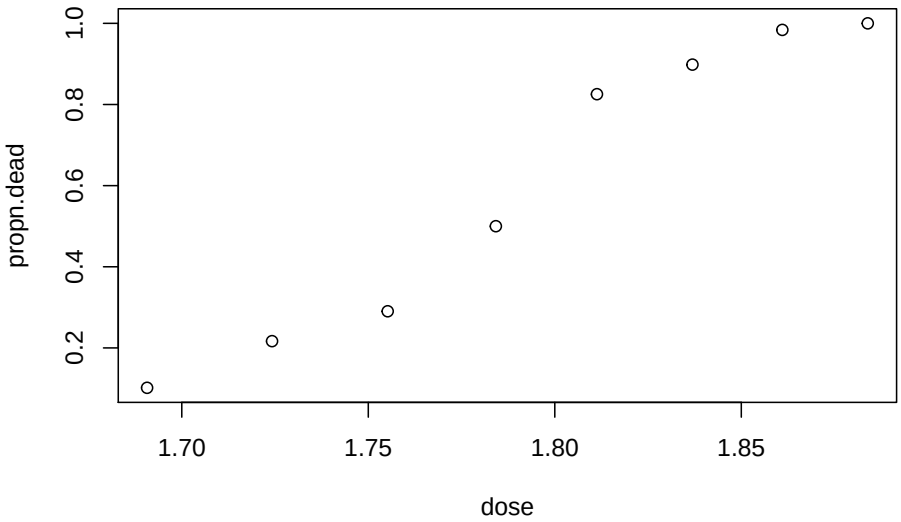
The research question is: does dose affect probability of killing insects? The first thing we probably want to do is calculate the proportions:

```
(beetle$propn.dead<-beetle$killed/beetle$number)
```

```
## [1] 0.1017 0.2167 0.2903 0.5000 0.8254 0.8983 0.9839
## [8] 1.0000
```

It's also reasonable to just plot the relationship between dose and proportion of deaths.

```
with(beetle,plot(dose,propn.dead))
```



Notice that the y-axis is by definition bounded between 0 and 1.

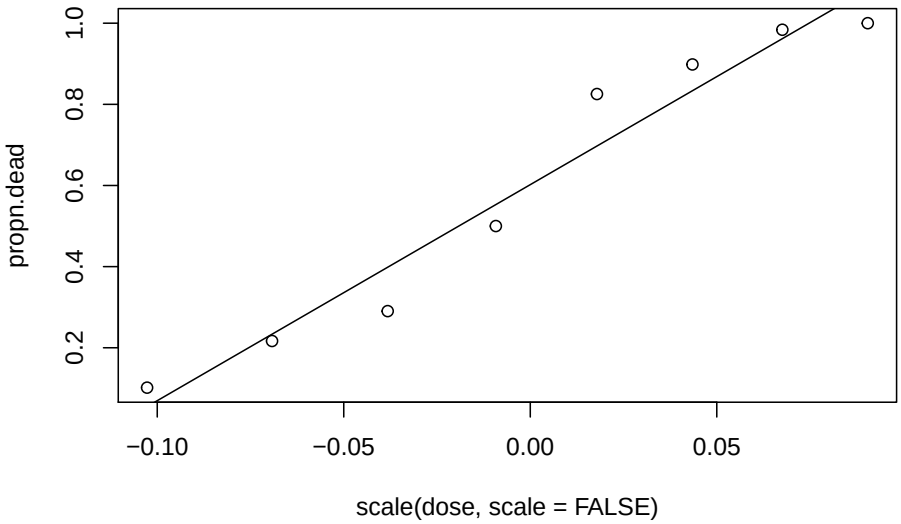
We could easily fit a linear model to this data-set. We may want to center the predictor, for reasons discussed earlier:

```
fm<-lm(propn.dead~scale(dose,scale=FALSE),beetle)
summary(fm)$coefficients
```

```
##                               Estimate Std. Error t value
## (Intercept)                      0.602    0.03065   19.64
## scale(dose, scale = FALSE)       5.325    0.48573   10.96
##                                Pr(>|t|)
## (Intercept)                   1.129e-06
## scale(dose, scale = FALSE) 3.422e-05
```

Next, add the best-fit (linear model) line to the plot:

```
with(beetle,plot(scale(dose,scale=FALSE),
                 propn.dead))
abline(coef(fm))
```



What's the interpretation of the coefficients? Since the predictor is centered, the intercept represents the mean proportion of beetles killed (0.62), and the slope represents the increase in the proportion of beetles killed when the dose is increased by one unit, from an initial dose of 0. The proportion of beetles killed with one unit increase in dose is larger than 1, which is obviously impossible for

a proportion. It is because of problems like these that a simple linear model does not suffice for such data.

Instead of using the linear model, we model log odds instead of proportions as a function of dose. Odds are defined as the ratio of the probability of success and the probability of failure (here, success is operationalized as the beetles being killed):

$$\frac{p}{1-p} \tag{5.35}$$

and taking the log will give us log odds.

We are going to model log odds (instead of probability) as a linear function of dose.

$$\log \frac{p}{1-p} = \beta_0 + \beta_1 \text{dose} \tag{5.36}$$

The model above is called the logistic regression model.

Once we have estimated the β parameters,
we can move back from log odds space to probability space using simple algebra.

Given a model like

$$\log \frac{p}{1-p} = \beta_0 + \beta_1 \text{dose} \tag{5.37}$$

If we exponentiate each side, we get:

$$\exp \log \frac{p}{1-p} = \frac{p}{1-p} = \exp(\beta_0 + \beta_1 \text{dose}) \tag{5.38}$$

So now we just solve for p, and get (check this):

$$p = \frac{\exp(\beta_0 + \beta_1 \text{dose})}{1 + \exp(\beta_0 + \beta_1 \text{dose})} \tag{5.39}$$

We fit the model in R as follows. Note that as long as we are willing to avoid interpreting the intercept and just interpret the estimate of β_1, there is no need to center the predictor here:

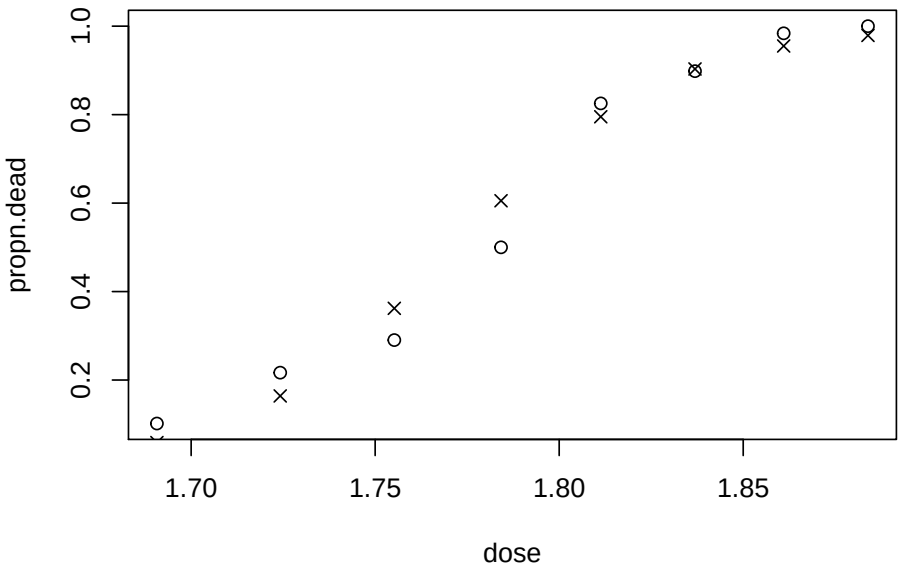
```
fm1<-glm(propn.dead~dose,
         family=binomial(logit),
         weights=number,
         data=beetle)
summary(fm1)
```

```
##
## Call:
## glm(formula = propn.dead ~ dose, family = binomial(logit), data = beetle,
##     weights = number)
##
## Deviance Residuals:
##    Min      1Q  Median      3Q     Max
## -1.594  -0.394   0.833   1.259   1.594
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept)   -60.72       5.18   -11.7   <2e-16
## dose           34.27       2.91    11.8   <2e-16
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 284.202  on 7  degrees of freedom
## Residual deviance:  11.232  on 6  degrees of freedom
## AIC: 41.43
##
## Number of Fisher Scoring iterations: 4
```

In the glm function call, `binomial(logit)` can be written simply as `binomial()`, as the logit link is the default. Other possible links are probit, cauchit, log, and cloglog—see the documentation in R for `family` for details.

We can also plot the observed proportions and the fitted values together; the fit looks pretty good.

```
plot(propn.dead~dose,beetle)
points(fm1$fitted~dose,beetle,pch=4)
```



We can now compute the log odds of death for concentration 1.7552 (for example):

```
## compute log odds of death for
## concentration 1.7552:
x<-as.matrix(c(1, 1.7552))
#log odds:
(log.odds<-t(x)%*%coef(fm1))
```

```
##          [,1]
## [1,] -0.5662
```

We can also obtain the variance-covariance matrix of the fitted coefficients, and use it to compute the variance-covariance matrix for the does 1.7552:

```
### compute CI for log odds:
## Get vcov matrix:
(vcovmat<-vcov(fm1))
```

```
##             (Intercept)    dose
## (Intercept)       26.84 -15.082
## dose             -15.08   8.481
```

```
## x^T VCOV x for dose 1.7552:
(var.log.odds<-t(x)%*%vcovmat%*%x)
```

```
##         [,1]
## [1,] 0.02168
```

Assuming that the log odds have a distribution that is approximately normal, we can compute the confidence interval for the log odds of death given dose 1.7552:

```
##lower
(lower<-log.odds-1.96*sqrt(var.log.odds))
```

```
##         [,1]
## [1,] -0.8548
```

```
##upper
(upper<-log.odds+1.96*sqrt(var.log.odds))
```

```
##         [,1]
## [1,] -0.2776
```

The lower and upper confidence interval bounds on the probability scale can be computed by using equation 5.39.

```
(meanprob<-exp(log.odds)/(1+exp(log.odds)))
```

```
##         [,1]
## [1,] 0.3621
```

```
(lowerprob<-exp(lower)/(1+exp(lower)))
```

```
##         [,1]
## [1,] 0.2984
```

```
(upperprob<-exp(upper)/(1+exp(upper)))
```

```
##       [,1]
## [1,] 0.431
```

So for dose 1.7552, the probability of death is 0.36, with 95% confidence intervals 0.30 and 0.43.

Note that one should not try to predict outside the range of the design matrix. For example, in the beetle data, the dose ranges from 1.69 to 1.88. We should not try to compute probabilities for dose 2.5, say, since we have no knowledge about whether the relationship remains unchanged beyond the upper bound of our design matrix.

5.2.5.2 Multiple logistic regression: Example from Hindi data}

We have eyetracking data from 10 participants reading Hindi sentences. We can compute skipping probability, the probability of skipping a word entirely (i.e., never fixating it). We first have to create a vector that has value 1 if the word has 0~ms total reading time, and 0 otherwise.

```
hindi10<-read.table("data/hindi10.txt",header=TRUE)
skip<-ifelse(hindi10$TFT==0,1,0)
hindi10$skip<-skip
## display relevant columns:
head(hindi10[,c(1,2,3,24,33,34)])
```

```
##   subj expt item word_complex SC skip
```

```
## 1   10 hnd1    6          0.0  1    1
## 2   10 hnd1    6          0.0  1    1
## 3   10 hnd1    6          0.0  2    0
## 4   10 hnd1    6          1.5  1    1
## 5   10 hnd1    6          0.0  1    1
## 6   10 hnd1    6          0.5  1    0
```

```
fm_skip<-glm(skip ~ word_complex+SC,family=binomial(),hindi10)
summary(fm_skip)
```

```
##
## Call:
## glm(formula = skip ~ word_complex + SC, family = binomial(),
##     data = hindi10)
##
## Deviance Residuals:
##    Min      1Q  Median      3Q     Max
## -1.100  -0.884  -0.674   1.256   2.682
##
## Coefficients:
##              Estimate Std. Error z value Pr(>|z|)
## (Intercept)   -0.1838     0.0266   -6.91  4.7e-12
## word_complex  -0.6291     0.0281  -22.38  < 2e-16
## SC            -0.5538     0.0224  -24.70  < 2e-16
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 31753  on 27065  degrees of freedom
## Residual deviance: 30492  on 27063  degrees of freedom
## AIC: 30498
##
## Number of Fisher Scoring iterations: 4
```

The above example also illustrates the second way to set up the data for logistic (multiple) regression: the dependent variable can simply be a 1,0 value instead of proportions. So, in the beetle data, you could recode the data to have 1s and 0s instead of proportions.

Assuming that you have recoded the column for status (dead or alive after exposure), the glm function call for the Beetle data-set would be:

```
glm(dead~dose,family=binomial(),beetle)
```

Note that logistic regression assumes independence of each data point; this assumption is violated in the Hindi data, because it has repeated measures data.

5.2.5.3 Deviance

We saw encountered deviance earlier in connection with ANOVA.

The deviance is more generally defined as

$$D = 2[logLik(\bar{x}; y) - logLik(mu_0; y)] \tag{5.40}$$

where $logLik(\bar{x}; y)$ is the log likelihood of the saturated model (the model with the maximal number of parameters that can be fit), and $logLik(mu_0; y)$ is the log likelihood of the model with parameter of interest having the null hypothesis value μ_0. As we saw earlier, D has a chi-squared distribution.

The deviance for the normal distribution is

$$D = \frac{1}{\sigma^2} \sum (y_i - \bar{y})^2 \tag{5.41}$$

[See p. 80 onwards in the Dobson et al book for proofs and more detail.]

Deviance for the binomial distribution is defined as $D = \sum d_i$, where:

$$d_i = -2 \times n_i [y_i \log \left(\frac{\widehat{\mu}_i}{y_i} \right) + (1 - y_i) \log \left(\frac{1 - \widehat{\mu}_i}{1 - y_i} \right)] \tag{5.42}$$

The basic idea here is that if the model fit is good, Deviance will

have a χ^2 distribution with $N - p$ degrees of freedom, where N is the number of data-points, and p the number of parameters. So that is what we will use for assessing model fit.

We will also use deviance for hypothesis testing. The difference in deviance (confusingly called residual deviance) between two models also has a χ^2 distribution (this should remind you of ANOVA), with dfs being $p - q$, where q is the number of parameters in the nukk model, and p the number of parameters in the full model.

We discuss hypothesis testing first, then evaluating goodness of fit using deviance.

5.2.5.3.1 Hypothesis testing: Residual deviance

Returning to our beetle data, let's say we fit our model:

```
glm1<-glm(propn.dead~dose,binomial(logit),
          weights=number,data=beetle)
```

The summary output shows us the number of iterations that led to the parameter estimates:

```
summary(glm1)
```

```
##
## Call:
## glm(formula = propn.dead ~ dose, family = binomial(logit), data = beetle,
##     weights = number)
##
## Deviance Residuals:
##    Min      1Q  Median      3Q     Max
## -1.594  -0.394   0.833   1.259   1.594
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept)   -60.72       5.18   -11.7   <2e-16
```

```
## dose            34.27       2.91    11.8   <2e-16
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 284.202  on 7  degrees of freedom
## Residual deviance:  11.232  on 6  degrees of freedom
## AIC: 41.43
##
## Number of Fisher Scoring iterations: 4
```

But we also see something called **Null deviance** and **Residual deviance**. These are used to evaluate quality of model fit. Recall that we can compute the fitted values and compare them to the observed values:

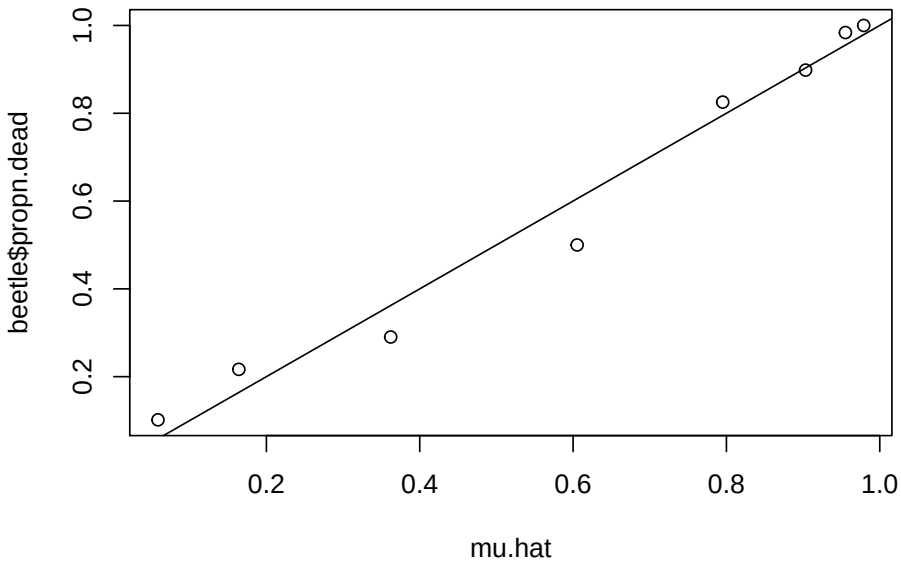
```
# beta.hat is (-60.71745 ,   34.27033)
(eta.hat<-  -60.71745 +   34.27033*beetle$dose)
```

```
## [1] -2.7766 -1.6285 -0.5662  0.4277  1.3564  2.2337
## [7]  3.0596  3.8444
```

```
(mu.hat<-exp(eta.hat)/(1+exp(eta.hat)))
```

```
## [1] 0.0586 0.1640 0.3621 0.6053 0.7952 0.9032 0.9552
## [8] 0.9790
```

```
# compare mu.hat with observed proportions
plot(mu.hat,beetle$propn.dead)
abline(0,1)
```



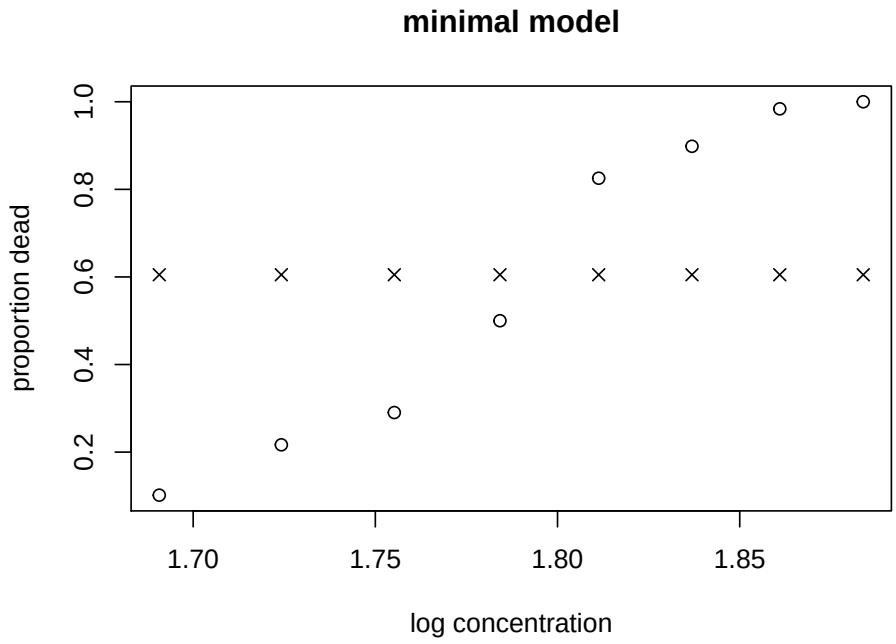
To evaluate whether dose has an effect, we will do something analogous to the model comparison methods we saw earlier. First, fit a model with only an intercept. Notice that the null deviance is 284 on 7 degrees of freedom.

```
null.glm<-glm(propn.dead~1,binomial(logit),
          weights=number,data=beetle)
summary(null.glm)
```

```
##
## Call:
## glm(formula = propn.dead ~ 1, family = binomial(logit), data = beetle,
##     weights = number)
##
## Deviance Residuals:
##    Min      1Q  Median      3Q     Max
##  -8.11   -5.29    1.10    5.62    7.77
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept)   0.4263     0.0933    4.57  4.9e-06
##
```

```
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 284.2  on 7  degrees of freedom
## Residual deviance: 284.2  on 7  degrees of freedom
## AIC: 312.4
##
## Number of Fisher Scoring iterations: 4
```

```
plot(beetle$dose,beetle$propn.dead,xlab="log concentration",
    ylab="proportion dead",main="minimal model")
points(beetle$dose,null.glm$fitted,pch=4)
```



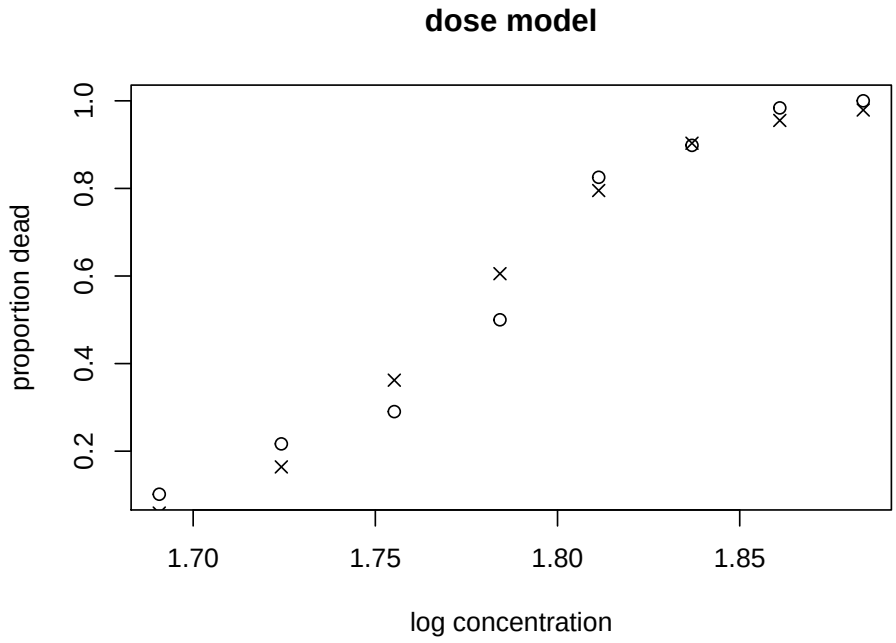
Add a term for dose. Now, the residual deviance is 11.2 on 6 dfs:

```
dose.glm<-glm(propn.dead~dose,binomial(logit),
          weights=number,data=beetle)
summary(dose.glm)
```

```
##
```

```
## Call:
## glm(formula = propn.dead ~ dose, family = binomial(logit), data = beetle,
##     weights = number)
##
## Deviance Residuals:
##    Min      1Q  Median      3Q     Max
## -1.594  -0.394   0.833   1.259   1.594
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept)   -60.72       5.18   -11.7   <2e-16
## dose           34.27       2.91    11.8   <2e-16
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 284.202  on 7  degrees of freedom
## Residual deviance:  11.232  on 6  degrees of freedom
## AIC: 41.43
##
## Number of Fisher Scoring iterations: 4
```

```
plot(beetle$dose,beetle$propn.dead,xlab="log concentration",
    ylab="proportion dead",main="dose model")
points(beetle$dose,dose.glm$fitted,pch=4)
```



The change in deviance from the null model is 284.2-11.2=273 on 1 df. Since the critical $\chi^2_1 = 3.84$, we reject the null hypothesis that $\beta_1 = 0$.

You can do the model comparison using the anova function. Note that no statistical test is calculated; you need to do that yourself.

```
anova(null.glm,dose.glm)
```

```
## Analysis of Deviance Table
##
## Model 1: propn.dead ~ 1
## Model 2: propn.dead ~ dose
##   Resid. Df Resid. Dev Df Deviance
## 1         7      284.2
## 2         6       11.2  1      273
```

Actually, you don't even need to define the null model; the anova function automatically compares the fitted model to the null model:

```
anova(dose.glm)
```

```
## Analysis of Deviance Table
##
## Model: binomial, link: logit
##
## Response: propn.dead
##
## Terms added sequentially (first to last)
##
##
##      Df Deviance Resid. Df Resid. Dev
## NULL                     7      284.2
## dose  1      273         6       11.2
```

5.2.5.3.2 Assessing goodness of fit of a fitted model

The deviance for a given degrees of freedom v should have a χ^2_v distribution for the model to be adequate. As an example, consider the null model above. The deviance is clearly much larger than the 95th percentile cutoff point of the chi-squared distribution with 7 dfs, so the model is not adequate.

```
deviance(null.glm)
```

```
## [1] 284.2
```

```
## critical value:
qchisq(0.95,df=7)
```

```
## [1] 14.07
```

Now consider the model with dose as predictor. The deviance is less than the 95th percentile, so the fit is adequate.

```
deviance(dose.glm)
```

```
## [1] 11.23
```

```
qchisq(0.95,df=6)
```

```
## [1] 12.59
```

5.2.5.3.3 Residuals in GLMs

In the binomial distribution, Deviance $D = \sum d_i$, where:

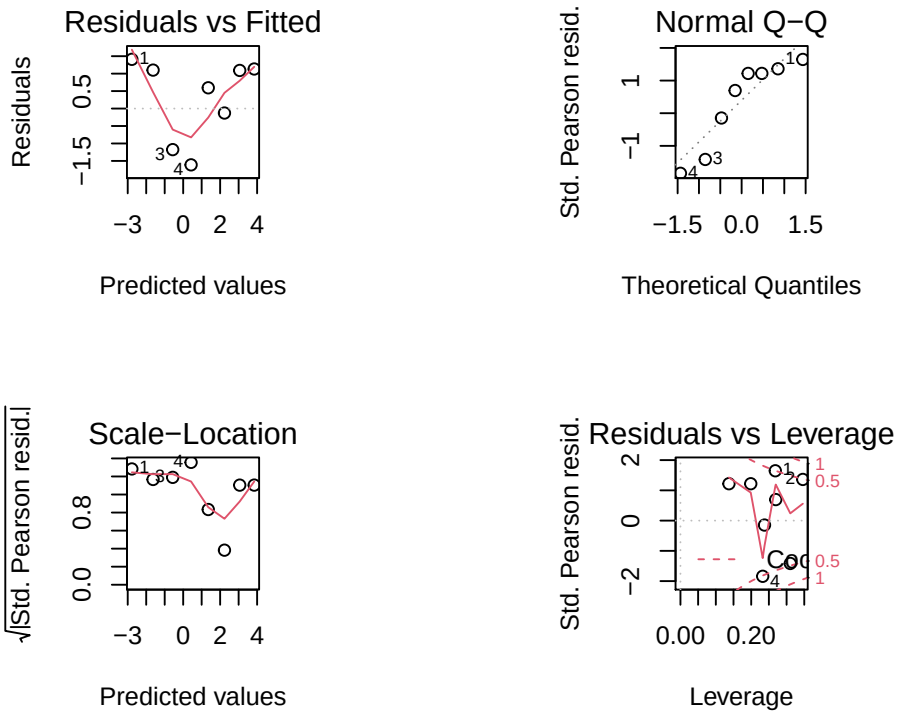
$$d_i = -2 \times n_i[y_i \log(\frac{\hat{\mu}_i}{y_i}) + (1 - y_i) \log(\frac{1 - \hat{\mu}_i}{1 - y_i})] \tag{5.43}$$

The i-th deviance residual is defined as:

$$e_{D,i} = sgn(y_i - \hat{\mu}_i) \times \sqrt{d_i} \tag{5.44}$$

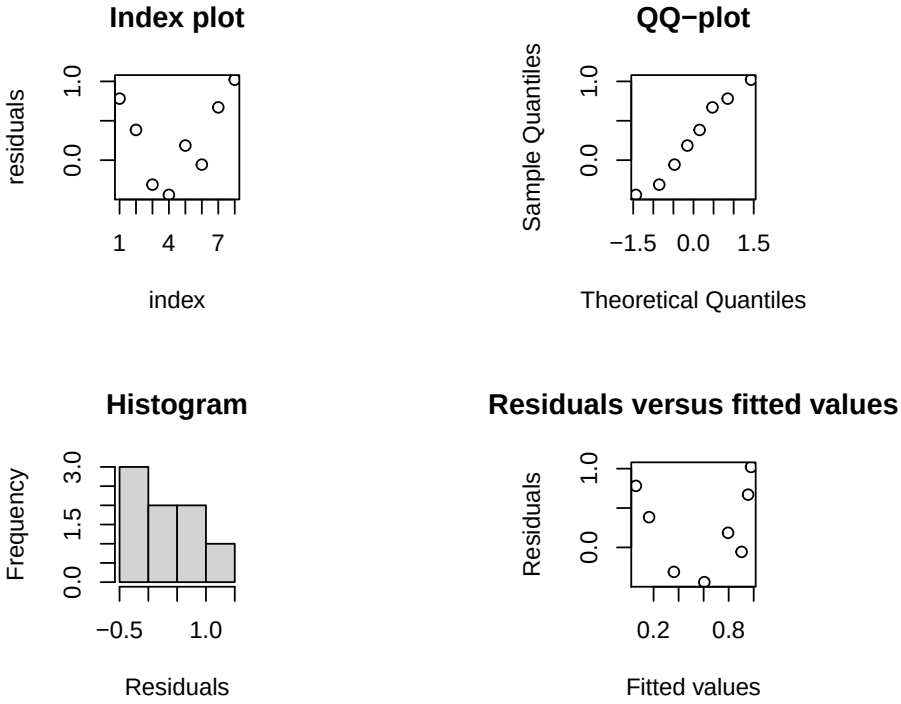
These can be used to check for model adequacy as discussed earlier in the context of linear models. One can just use the plot function inspect the residuals:

```
op<-par(mfrow=c(2,2),pty="s")
plot(dose.glm)
```



Alternatively, one can do this by hand:

```
op<- par(mfrow=c(2,2),pty="s")
plot(dose.glm$resid,
     xlab="index",ylab="residuals",main="Index plot")
qqnorm(dose.glm$resid,main="QQ-plot")
hist(dose.glm$resid,xlab="Residuals",main="Histogram")
plot(dose.glm$fit,dose.glm$resid,xlab="Fitted values",
     ylab="Residuals",
     main="Residuals versus fitted values")
```



5.3 Exercises

5.3.1 Estimating the parameters in a linear model

The data consist of a repeated measures experiment comparing two conditions which are labeled Type 1 and Type 2. The column Sub refers to subject id, and the column ID refers to item id. RT refers to reading time in seconds (we have converted it below to milliseconds); NA is missing data. You can ignore the other columns. This is a standard Latin square design.

```
## load data:
dat<-read.csv("data/Type1Type2_data.csv",header=TRUE)
## convert RT to milliseconds:
dat$RT<-dat$RT*1000
## choose critical region:
```

```
word_n<-4
## subset critical data:
crit<-subset(dat,Position==word_n)
```

Using 0,1 contrast coding (treatment contrast coding), fit a simple linear model (**not** a linear mixed model!) to RT, with Type as a predictor. This model is incorrect for the data but we ignore this detail for now. Notice that there is missing data in this data-set; you have to deal with that complication somehow.

- Extract the model/design matrix X from the fitted model.
- Extract sigma estimate $\hat{\sigma}$.
- Compute $\hat{\sigma}(X^T X)^{-1}$.
- Use the X matrix and the y vector to estimate the $\hat{\beta}$ vector.
- Compute the variance covariance matrix of $\hat{\beta}$.
- Display the bivariate distribution of the sampling distribution of the intercept $\hat{\beta}_0$ and slope $\hat{\beta}_1$.
- What is the correlation between $\hat{\beta}_0$ and $\hat{\beta}_1$?

Next, using ± 1 contrast coding (sum contrast coding), fit a simple linear model (**not** a linear mixed model!) to RT, with Type as a predictor. This model is incorrect for the data but we ignore this detail for now.

- Extract the model/design matrix X from the fitted model.
- Extract sigma estimate $\hat{\sigma}$.
- Compute $\hat{\sigma}(X^T X)^{-1}$.
- Use the X matrix and the y vector to estimate the $\hat{\beta}$ vector.
- Compute the variance covariance matrix of $\hat{\beta}$.
- Display the bivariate distribution of the sampling distribution of the intercept $\hat{\beta}_0$ and slope $\hat{\beta}_1$ as on slide 33 of the matrix formulation lecture.
- What is the correlation between $\hat{\beta}_0$ and $\hat{\beta}_1$?
- Use the likelihood ratio test (use the anova() function in R), find out if we can reject the null hypothesis that Type 1 and 2 have no difference in reading time. Is the result of the likelihood ratio test any different from that in Question 1?

- Speculate on why there a difference between the correlations of the $\hat{\beta}_0$ and $\hat{\beta}_1$ in the treatment vs. sum contrast coding you carried out above.

5.3.2 Using ANOVA to carry out hypothesis testing

Using the above data, with the sum contrast coding you defined above, carry out a null hypothesis significance test using the anova() function with an appropriate linear mixed model. What null hypothesis are you testing here, and what is your conclusion?

5.3.3 Computing ANOVA by hand

In this chapter, we saw how the ANOVA is computed in R:

```
m1<-lm(y~x_c)
m0<-lm(y~1)
anova(m0,m1)
```

We also saw how an ANOVA is standardly summarized using matrix notation:

TABLE 5.2: The standard way to summarize an ANOVA analysis.

Source of variance	df	Sum of squares	Mean square
Model with β_0	q	$\hat{\beta}_0^T X_0^T Y$	
Improvement due to β_1	p-q	$\hat{\beta}_1 X_1^T Y - \hat{\beta}_0^T X_0^T Y$	$\frac{\hat{\beta}_1 X_1^T Y - \hat{\beta}_0^T X_0^T Y}{p-q}$
Residual	n-p	$Y^T Y - \hat{\beta}_1^T X_1^T Y$	$\frac{Y^T Y - \hat{\beta}_1^T X_1^T Y}{n-p}$
Total	n	$Y^T Y$	

Reproduce this table for the analysis shown above as R code, and show that the results of `anova()` function (the F-value) above are identical to those produced through this table.

5.3.4 Generalized linear (mixed) model

In the chapter, we saw the following code in connection with the logistic regression model. This is Hindi eyetracking data excerpted from Husain et al. (2015). The code below evaluates the effect of various predictors of sentence processing difficulty, such as word complexity and storage cost (SC), on skipping probability.

```
hindi10<-read.table("data/hindi10.txt",header=TRUE)
skip<-ifelse(hindi10$TFT==0,1,0)
hindi10$skip<-skip
## display relevant columns:
head(hindi10[,c(1,2,3,24,33,34)])
```

```
##   subj expt item word_complex SC skip
## 1   10 hnd1    6          0.0  1    1
## 2   10 hnd1    6          0.0  1    1
## 3   10 hnd1    6          0.0  2    0
## 4   10 hnd1    6          1.5  1    1
## 5   10 hnd1    6          0.0  1    1
## 6   10 hnd1    6          0.5  1    0
```

```
#fm_skip<-glm(skip ~ word_complex+SC,family=binomial(),hindi10)
```

Fit a linear mixed model using the function `glmer` and the link function `family=binomial()`, with varying intercepts for subject and item, as well as varying slopes if possible, with centered word complexity and storage cost as predictors, and skipping probability (represented as 0,1 values) as dependent measure. Using the anova function, find out whether there is evidence for (a) word complexity, and (b) storage cost affecting skipping probability. Display the effects of these two variables on skipping probability on the probability scale by back-transforming the mean and 95% confidence interval of each effect from the log odds scale to the probability scale.

6

Contrast coding

Whenever one uses a categorical factor as a predictor in a linear (mixed) model—for example when testing the difference in a dependent variable between two or three experimental conditions—then it is necessary to code the discrete factor levels into numeric predictor variables. This coding is termed *contrast coding*. For example, in the linear modeling chapter, we coded two experimental conditions as -1 and $+1$, i.e., implementing a sum contrast. Those *contrasts* are the numbers that we give to numeric predictor variables to encode specific hypotheses about differences between factor levels and to create predictor terms to test these hypotheses in linear models, including Bayesian linear (mixed) models.

This chapter will introduce contrast coding. The descriptions are in large parts taken from Schad et al. (2020b) (which is published under a CC-BY license) and adapted for the current chapter.

Consider a situation where we want to test differences in a dependent variable between three factor levels. An example could be differences in response times between three levels of word class (noun, verb, adjective). We might be interested in whether word class influences response times. In frequentist statistics, one way to approach this question would be to run an ANOVA and compute an omnibus F-test for whether word class explains response times. However, if based on such omnibus approaches we find support for an influence of word class on response times, it remains unclear where this effect actually comes from, i.e., whether it originated from the nouns, verbs, or adjectives. However, scientists typically have a priori expectations about which groups differ from each other. In this chapter, we will show how to test specific hypotheses directly, which gives a lot of control over the analyses.

Specifically, we show how planned comparisons between specific conditions (groups) or clusters of conditions, are implemented as contrasts. This is a very effective way to align expectations with the statistical model.

6.1 Basic concepts illustrated using a two-level factor

We first consider the simplest case: suppose we want to compare the means of a dependent variable (DV) such as response times between two groups of subjects. R can be used to simulate data for such an example. Such simulated data is available in the R-package `bcogsci` (to install this package, see: https://github.com/bnicenboim/bcogsci) as the data set `df_contrasts1`. The simulations assumed longer response times in condition F1 ($\mu_1 =$ 0.8 sec) than F2 ($\mu_2 = 0.4$ sec). The data from the 10 simulated subjects are aggregated and summary statistics are computed for the two groups.

```
library(bcogsci)
data("df_contrasts1")
df_contrasts1
```

```
## # A tibble: 10 x 3
##    F        DV    id
##    <fct> <dbl> <int>
##  1 F1    0.636     1
##  2 F1    0.841     2
##  3 F1    0.555     3
##  4 F1    1.03      4
##  5 F1    0.938     5
##  6 F2    0.123     6
##  7 F2    0.304     7
##  8 F2    0.659     8
##  9 F2    0.469     9
## 10 F2    0.444    10
```

```
str(df_contrasts1)
```

```
## tibble[,3] [10 x 3] (S3: tbl_df/tbl/data.frame)
##  $ F : Factor w/ 2 levels "F1","F2": 1 1 1 1 1 2 2 2 2 2
##  $ DV: num [1:10] 0.636 0.841 0.555 1.029 0.938 ...
##  $ id: int [1:10] 1 2 3 4 5 6 7 8 9 10
```

```
## [1] 0.6
```

```
kbl(table1a, position="b", digits=1,
    caption="Summary statistics per condition for the simulated data.")
```

The results, displayed in Figure 6.1 and shown in Table 6.1, show that the assumed true condition means are exactly realized with the simulated data. The numbers are exact because the used `mvrnorm()` function (see `df_contrasts1`) ensures that the data are generated so that the sample mean yields the true means for each level. In real data-sets, of course, the sample means will vary from experiment to experiment.

A simple Bayesian linear model of `DV` on `F` using the function `brm` yields a straightforward estimate of the difference between the group means. We use rather vague priors. The estimates for the fixed effects are presented below:

```
fit_F <- lm(DV ~ 1 + F,
                 data = df_contrasts1)
```

TABLE 6.1: Summary statistics per condition for the simulated data.

Factor	N data	Est. means	Std. dev.	Std. errors
F1	5	0.8	0.2	0.1
F2	5	0.4	0.2	0.1

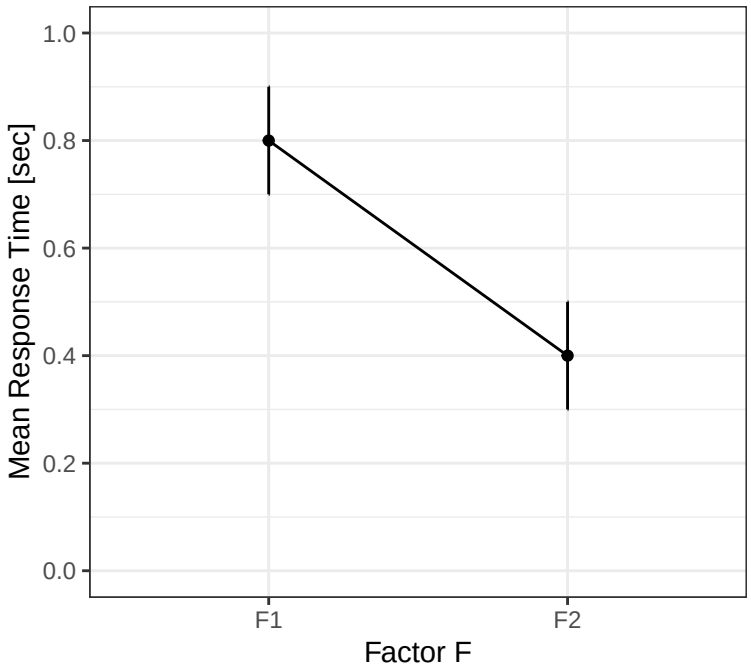


FIGURE 6.1: Means and standard errors of the simulated dependent variable (e.g., response times in seconds) in two conditions F1 and F2.

```
round(summary(fit_F)$coefficients,3)
```

```
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)      0.8      0.089   8.944    0.000
## FF2             -0.4      0.126  -3.162    0.013
```

Comparing the means for each condition with the coefficients (*Estimates*) reveals that (i) the intercept (0.8) is the mean for condition F1, $\widehat{\mu}_1$; and (ii) the slope (`FF2`: -0.4) is the difference between the true means for the two groups, $\widehat{\mu}_2 - \widehat{\mu}_1$ (Bolker, 2018):

$$\begin{array}{lll} \text{Intercept} = & \hat{\mu}_1 & = \text{estimated mean for F1} \\ \text{Slope (FF2)} = & \hat{\mu}_2 - \hat{\mu}_1 & = \text{estim. mean for F2} - \text{estim. mean for F1} \end{array} \tag{6.1}$$

The new information is the standard error for the difference between the two groups, and the t- and p-values corresponding to the null hypothesis test of no difference.

6.1.1 Default contrast coding: Treatment contrasts

How does the function `lm` arrive at these particular values for the intercept and slope? That is, why does the intercept assess the mean of condition `F1` and how do we know the slope measures the difference in means between `F2`—`F1`? This result is a consequence of the default contrast coding of the factor `F`. R assigns treatment contrasts to factors and orders their levels alphabetically. The first factor level (here: `F1`) is coded as 0 and the second level (here: `F2`) is coded as 1. This becomes clear when we inspect the current contrast attribute of the factor using the `contrasts` command:

```
contrasts(df_contrasts1$F)
```

```
##    F2
## F1  0
## F2  1
```

Why does this contrast coding yield these particular regression coefficients? Let's take a look at the regression equation. Let β_0 represent the intercept, and β_1 the slope. Then, the simple regression above expresses the belief that the expected response time y is a linear function of the factor `F`. In a more general formulation, this is written as follows: y is a linear function of some predictor x with regression coefficients for the intercept, β_0, and for the factor, β_1:

$$y = \beta_0 + \beta_1 x \tag{6.2}$$

This equation is part of the likelihood in a statistical model. So, if $x = 0$ (condition `F1`), y is $\beta_0 + \beta_1 \cdot 0 = \beta_0$; and if $x = 1$ (condition `F2`), y is $\beta_0 + \beta_1 \cdot 1 = \beta_0 + \beta_1$.

Expressing the above in terms of the estimated coefficients:

$$\begin{array}{llll} \text{estim. value for F1} = & \hat{\mu}_1 = & \hat{\beta}_0 = & \text{Intercept} \\ \text{estim. value for F2} = & \hat{\mu}_2 = & \hat{\beta}_0 + \hat{\beta}_1 = & \text{Intercept} + \text{Slope} \end{array} \quad \text{(FF2)} \tag{6.3}$$

It is useful to think of such unstandardized regression coefficients as difference scores; they express the increase in the dependent variable y associated with a change in the independent variable x of 1 unit, such as going from 0 to 1 in this example. The difference between condition means is $0.4 - 0.8 = -0.4$, which is the estimated regression coefficient $\hat{\beta}_1$. The sign of the slope is negative because we have chosen to subtract the larger mean `F1` score from the smaller mean `F2` score.

6.1.2 Defining hypotheses

The analysis of the regression equation demonstrates that in the treatment contrast the intercept assesses the average response in the baseline condition, whereas the slope estimates the difference between condition means. However, these are just verbal descriptions of what each coefficient assesses. Is it also possible to formally write down what each coefficient assesses? Moreover, is it possible to relate this to formal null hypotheses that are encoded in each of these two coefficients? The relation to null hypothesis tests is directly possible in frequentist statistics.

From the perspective of parameter estimation and formal hypothesis tests, the slope represents the main test of interest, so we consider this first. The treatment contrast specifies that the slope β_1 estimates the difference in means between the two levels of the factor F. This can formally be written as:

$$\beta_1 = \mu_{F2} - \mu_{F1} \tag{6.4}$$

or equivalently:

$$\beta_1 = -1 \cdot \mu_{F1} + 1 \cdot \mu_{F2} \tag{6.5}$$

This can express the null hypothesis that the difference in means between the two levels of the factor F is 0; formally, the null hypothesis H_0 is that $H_0 : \beta_1 = 0$:

$$H_0 : \beta_1 = \mu_{F2} - \mu_{F1} = 0 \tag{6.6}$$

or equivalently:

$$H_0 : \beta_1 = -1 \cdot \mu_{F1} + 1 \cdot \mu_{F2} = 0 \tag{6.7}$$

The ± 1 weights in the parameter estimation and null hypothesis statements directly express which means are compared by the treatment contrast.

The intercept in the treatment contrast estimates a quantity and expresses a null hypothesis that is usually of little interest: it estimates the mean in condition F1, and can be used to test whether this mean of F1 is 0. Formally, the parameter β_0 estimates the following quantity:

$$\beta_0 = \mu_{F1} \tag{6.8}$$

or equivalently:

$$\beta_0 = 1 \cdot \mu_{F1} + 0 \cdot \mu_{F2}. \tag{6.9}$$

This can also be written as a formal null hypothesis, which is $H_0 : \beta_0 = 0$:

$$H_0 : \beta_0 = \mu_{F1} = 0 \tag{6.10}$$

or equivalently:

$$H_0 : \beta_0 = 1 \cdot \mu_{F1} + 0 \cdot \mu_{F2} = 0. \tag{6.11}$$

The fact that the intercept term formally tests the null hypothesis that the mean of condition `F1` is zero is in line with our previous derivation (see equation 6.1).

In R, factor levels are ordered alphabetically and by default the first level is used as the baseline in treatment contrasts. Obviously, this default mapping will only be correct for a given data-set if the levels' alphabetical ordering matches the desired contrast coding. When it does not, it is possible to re-order the levels. Here is one way of re-ordering the levels in R:

```
df_contrasts1$Fb <- factor(df_contrasts1$F,  levels = c("F2","F1"))
contrasts(df_contrasts1$Fb)
```

```
##    F1
## F2  0
## F1  1
```

This re-ordering did not change any data associated with the factor, only one of its attributes. With this new contrast attribute a simple linear model yields the following result.

```
fit_Fb <- lm(DV ~ 1 + Fb,
                 data = df_contrasts1)
```

```
round(summary(fit_Fb)$coefficients,3)
```

```
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)      0.4      0.089   4.472    0.002
## FbF1             0.4      0.126   3.162    0.013
```

The model now estimates different quantities. The intercept now

codes the mean of condition F2, and the slope measures the difference in means between F1 minus F2. This represents an alternative coding of the treatment contrast.

6.1.3 Sum contrasts

Treatment contrasts are only one of many options. It is also possible to use sum contrasts, which code one of the conditions as -1 and the other as $+1$, effectively 'centering' the effects at the grand mean (GM, i.e., the mean of the two group means). Here, we rescale the contrast to values of -0.5 and $+0.5$, which makes the estimated treatment effect the same as for treatment coding and easier to interpret.

To use this contrast in a linear regression, use the `contrasts` function:

```
(contrasts(df_contrasts1$F) <- c(-0.5,+0.5))
fit_mSum <- lm(DV ~ 1 + F,
               data = df_contrasts1)
```

```
round(summary(fit_mSum)$coefficients,3)
```

```
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)      0.6      0.063   9.487    0.000
## F1              -0.4      0.126  -3.162    0.013
```

Here, the slope (F1) again codes the difference of the groups associated with the first and second factor levels. It has the same value as in the treatment contrast. However, the intercept now represents the estimate of the average of condition means for F1 and F2, that is, the GM. This differs from the treatment contrast. For the scaled sum contrast:

$$\begin{aligned} \text{Intercept} = \quad & (\hat{\mu}_1 + \hat{\mu}_2)/2 \quad && = \text{estimated mean of F1 and F2} \\ \text{Slope (F1)} = \quad & \hat{\mu}_2 - \hat{\mu}_1 \quad && = \text{estim. mean for F2} - \text{estim. mean for F1} \end{aligned} \tag{6.12}$$

Why does the intercept assess the GM and why does the slope test the group difference? This is the result of rescaling the sum contrast. The first factor level (`F1`) was coded as -0.5, and the second factor level (`F1`) as $+0.5$:

```
contrasts(df_contrasts1$F)
```

```
##    [,1]
## F1 -0.5
## F2  0.5
```

Let's again look at the regression equation to better understand what computations are performed. Again, β_0 represents the intercept, β_1 represents the slope, and the predictor variable x represents the factor `F`. The regression equation is written as:

$$y = \beta_0 + \beta_1 x \tag{6.13}$$

The group of `F1` subjects is then coded as -0.5, and the response time for the group of `F1` subjects is $\beta_0 + \beta_1 \cdot x_1 = 0.6 + (-0.4) \cdot (-0.5) = 0.8$. The `F2` group, to the contrary, is coded as $+0.5$. By implication, the mean of the `F2` group must be $\beta_0 + \beta_1 \cdot x_1 = 0.6 + (-0.4) \cdot 0.5 = 0.4$. Expressed in terms of the estimated coefficients:

$$\begin{aligned} \text{estim. value for F1} = \quad \hat{\mu}_1 = \quad \hat{\beta}_0 - 0.5 \cdot \hat{\beta}_1 = \quad \text{Intercept} - 0.5 \cdot \text{Slope (F1)} \\ \text{estim. value for F2} = \quad \hat{\mu}_2 = \quad \hat{\beta}_0 + 0.5 \cdot \hat{\beta}_1 = \quad \text{Intercept} + 0.5 \cdot \text{Slope (F1)} \end{aligned} \tag{6.14}$$

The unstandardized regression coefficient is a difference score: Taking a step of one unit on the predictor variable x, e.g., from -0.5 to $+0.5$, reflects a step from condition F1 to F2, and changes the dependent variable from 0.8 (for condition F1) to 0.4 (condition F2), giving us a difference of $0.4 - 0.8 = -0.4$. This is again the estimated regression coefficient $\hat{\beta}_1$. Moreover, as mentioned above, the intercept now assesses the GM of conditions F1 and F2: it is in the middle between condition means for F1 and F2.

So far we gave verbal statements about what is tested by the intercept and the slope in the case of the scaled sum contrast. It is possible to write these statements as formal parameter estimates and formal null hypotheses that are tested by each regression coefficient. In sum contrasts, the slope parameter β_1 assesses the following quantity:

$$\beta_1 = -1 \cdot \mu_{F1} + 1 \cdot \mu_{F2} \tag{6.15}$$

This can be formulated into the null hypothesis that the difference in means between the two levels of factor F is 0; formally, the null hypothesis H_0 is that

$$H_0 : \beta_1 = -1 \cdot \mu_{F1} + 1 \cdot \mu_{F2} = 0 \tag{6.16}$$

This estimates the same quantity and tests the same null hypothesis as the slope in the treatment contrast. The intercept, however, now assess a different quantity and expresses a different hypothesis about the data: it estimates the average of the two conditions F1 and F2, and tests the null hypothesis that this average is 0:

$$\beta_0 = 1/2 \cdot \mu_{F1} + 1/2 \cdot \mu_{F2} = \frac{\mu_{F1} + \mu_{F2}}{2} \tag{6.17}$$

And for the null hypothesis:

$$H_0 : \beta_0 = 1/2 \cdot \mu_{F1} + 1/2 \cdot \mu_{F2} = \frac{\mu_{F1} + \mu_{F2}}{2} = 0 \tag{6.18}$$

In balanced data, i.e., in data-sets where there are no missing data points, the average of the two conditions F1 and F2 is the GM. In unbalanced data-sets, where there are missing values, this average is the weighted GM. To illustrate this point, consider an example with fully balanced data and two equal group sizes of 5 subjects for each group F1 and F2. Here, the GM is also the mean across all subjects. Next, consider a highly simplified unbalanced data-set, where in condition F1 two observations of the dependent variable

are available with values of 2 and 3, and where in condition F2 only one observation of the dependent variables is available with a value of 4. In this data-set, the mean across all subjects is $\frac{2+3+4}{3} = \frac{9}{3} = 3$. However, the (weighted) GM as assessed in the intercept in a model using sum contrasts for factor F would first compute the mean for each group separately (i.e., $\frac{2+3}{2} = 2.5$, and 4), and then compute the mean across conditions $\frac{2.5+4}{2} = \frac{6.5}{2} = 3.25$. The GM of 3.25 is different from the mean across subjects of 3.

To summarize, treatment contrasts and sum contrasts are two possible ways to parameterize the difference between two groups; they estimate different quantities and test different hypotheses (there are cases, however, where the estimates / hypotheses are equivalent). Treatment contrasts compare one or more means against a baseline condition, whereas sum contrasts allow us to determine whether we can reject the null hypothesis that a condition's mean is the same as the GM (which in the two-group case also implies a hypothesis test that the two group means are the same). One question that comes up here is how one knows or formally derives what quantities are estimated by a given set of contrasts, and what hypotheses it can be used to test. This question will be discussed in detail below for the general case of any arbitrary contrasts.

6.1.4 Cell means parameterization and posterior comparisons

One alternative option is to use the so-called cell means parameterization. In this approach, one does not estimate an intercept term, and then differences between factor levels. Instead, each degree of freedom in a design is used to simply estimate the mean of one of the factor levels. As a consequence, no comparisons between condition means are estimated, but simply the mean of each experimental condition is estimated. Cell means parameterization is specified by explicitly removing the intercept term (which is added automatically in R by default) by adding a -1 in the regression formula:

```
fit_mCM <- lm(DV ~ -1 + F,
                  data = df_contrasts1)
```

```
round(summary(fit_mCM)$coefficients,3)
```

```
##     Estimate Std. Error t value Pr(>|t|)
## FF1      0.8      0.089   8.944    0.000
## FF2      0.4      0.089   4.472    0.002
```

Now, the regression coefficients (see the column labeled 'Estimate') estimate the mean of the first factor level (0.8) and the mean of the second factor level (0.4). This cell means parameterization usually does not allow a test of the hypotheses of interest, as these hypotheses usually relate to differences between conditions rather than to whether each condition differs from zero.

6.2 The hypothesis matrix illustrated with a three-level factor

Consider an example with the three word classes nouns, verbs, and adjectives. We load simulated data from a lexical decision task with response times as dependent variable. The research question is: do response times differ as a function of the between-subject factor word class with three levels: nouns, verbs, and adjectives? We here make the ad-hoc assumption that nouns may have longer response times and that adjectives may have shorter response times. Here, we specify word class as a between-subject factor. In cognitive science experiments, word class will usually vary within subjects and between items. However, the within- or between-subjects status of an effect is independent of its contrast coding; we assume the manipulation to be between subjects for ease of exposition. The concepts presented here extend to repeated measures designs that

are often analyzed using hierarchical Bayesian (linear mixed) models.

The following R code loads and displays the simulated the data.

```
data("df_contrasts2")
head(df_contrasts2)
```

```
## # A tibble: 6 x 3
##   F        DV    id
##   <fct> <int> <int>
## 1 nouns   476     1
## 2 nouns   517     2
## 3 nouns   491     3
## 4 nouns   516     4
## 5 verbs   464     5
## 6 verbs   439     6
```

As shown in Table 6.2, the estimated means reflect our assumptions about the true means in the data simulation: Response times are longest for nouns and shortest for adjectives. In the following sections, we use this data-set to illustrate sum. Furthermore, we will use an additional data set to illustrate repeated, polynomial, and custom contrasts. In practice, usually only one set of contrasts is selected when the expected pattern of means is formulated during the design of the experiment.

TABLE 6.2: Summary statistics per condition for the simulated data.

Factor	N data	Est. means	Std. dev.	Std. errors
adjectives	4	400.2	19.9	9.9
nouns	4	500.0	20.0	10.0
verbs	4	450.2	20.0	10.0

6.2.1 Sum contrasts

For didactic purposes, the next sections describe sum contrasts. Suppose that the expectation is that nouns are responded to slower and verbs words are responded to faster than the GM response time. Then, the research question could be: Do nouns differ from the GM and do adjectives differ from the GM? And if so, are they above or below the GM? We want to estimate the following two quantities:

$$\beta_1 = \mu_1 - \frac{\mu_1 + \mu_2 + \mu_3}{3} = \mu_1 - GM \tag{6.19}$$

and

$$\beta_2 = \mu_2 - \frac{\mu_1 + \mu_2 + \mu_3}{3} = \mu_2 - GM \tag{6.20}$$

This translates into the following two hypotheses:

$$H_{0_1} : \mu_1 = \frac{\mu_1 + \mu_2 + \mu_3}{3} = GM \tag{6.21}$$

and

$$H_{0_2} : \mu_2 = \frac{\mu_1 + \mu_2 + \mu_3}{3} = GM \tag{6.22}$$

H_{0_1} can also be written as:

$$\mu_1 = \frac{\mu_1 + \mu_2 + \mu_3}{3} \tag{6.23}$$

$$\Leftrightarrow \mu_1 - \frac{\mu_1 + \mu_2 + \mu_3}{3} = 0 \tag{6.24}$$

$$\Leftrightarrow \frac{2}{3}\mu_1 - \frac{1}{3}\mu_2 - \frac{1}{3}\mu_3 = 0 \tag{6.25}$$

This corresponds to estimating the quantity $\beta_1 = \frac{2}{3}\mu_1 - \frac{1}{3}\mu_2 - \frac{1}{3}\mu_3$.

Here, the weights $2/3, -1/3, -1/3$ are informative about how to combine the condition means to estimate the linear model coefficient and to define the null hypothesis.

H_{0_2} is also rewritten as:

$$\mu_2 = \frac{\mu_1 + \mu_2 + \mu_3}{3} \tag{6.26}$$

$$\Leftrightarrow \mu_2 - \frac{\mu_1 + \mu_2 + \mu_3}{3} = 0 \tag{6.27}$$

$$\Leftrightarrow -\frac{1}{3}\mu_1 + \frac{2}{3}\mu_2 - \frac{1}{3}\mu_3 = 0 \tag{6.28}$$

This corresponds to estimating the quantity $\beta_2 = -\frac{1}{3}\mu_1 + \frac{2}{3}\mu_2 - \frac{1}{3}\mu_3$. Here, the weights are $-1/3, 2/3, -1/3$, and they again indicate how to combine the condition means for estimating the regression coefficient and for defining the null hypothesis.

6.2.2 The hypothesis matrix

The weights of the condition means are not only useful to define parameter estimates and hypotheses. They also provide the starting step in a very powerful method which allows the researcher to generate the contrasts that are needed to test these hypotheses in a linear model. That is, what we did so far is to explain some kinds of different contrast codings that exist and what the hypotheses are that they test. That is, if a certain data-set is given and the goal is to estimate certain comparisons (and test certain hypotheses), then the procedure would be to check whether any of the contrasts that we encountered above happen to estimate these comparisons and test exactly the hypotheses of interest. Sometimes it suffices to use one of these existing contrasts. However, at other times, our research questions may not correspond exactly to any of the contrasts in the default set of standard contrasts provided in R. For these cases, or simply for more complex designs, it is very useful to know how contrast matrices are created. Indeed, a relatively simple procedure exists in which we write our comparisons or hypotheses formally, extract the weights of the condition means

from the comparisons/hypotheses, and then automatically generate the correct contrast matrix that we need in order to estimate these comparisons or test these hypotheses in a linear model. Using this powerful method, it is not necessary to find a match to a contrast matrix provided by the family of functions in R starting with the prefix `contr`. Instead, it is possible to simply define the comparisons that one wants to estimate or the hypotheses that one wants to test, and to obtain the correct contrast matrix for these in an automatic procedure. Here, for pedagogical reasons, we show some examples of how to apply this procedure in cases where the comparisons/hypotheses *do* correspond to some of the existing contrasts.

Defining a custom contrast matrix involves four steps:

1. Write down the estimated comparisons or hypotheses
2. Extract the weights and write them into what we will call a *hypothesis matrix*
3. Apply the *generalized matrix inverse* to the hypothesis matrix to create the contrast matrix
4. Assign the contrast matrix to the factor and run the linear (mixed) model

Let us apply this four-step procedure to our example of the sum contrast. The first step, writing down the estimated parameters and hypotheses, is shown above. The second step involves writing down the weights that each comparison / hypothesis gives to condition means. The weights for the first comparison or null hypothesis are `wH01=c(+2/3, -1/3, -1/3)`, and the weights for the second comparison or null hypothesis are `wH02=c(-1/3, +2/3, -1/3)`.

Before writing these into a hypothesis matrix, we also define the estimated quantity and the null hypothesis for the intercept term. The intercept parameter estimates the mean across all conditions:

$$\beta_0 = \frac{\mu_1 + \mu_2 + \mu_3}{3} \tag{6.29}$$

$$\beta_0 = \frac{1}{3}\mu_1 + \frac{1}{3}\mu_2 + \frac{1}{3}\mu_3 \tag{6.30}$$

This corresponds to the null hypothesis that the mean across all conditions is zero:

$$H_{0_0} : \frac{\mu_1 + \mu_2 + \mu_3}{3} = 0 \tag{6.31}$$

$$H_{0_0} : \frac{1}{3}\mu_1 + \frac{1}{3}\mu_2 + \frac{1}{3}\mu_3 = 0 \tag{6.32}$$

This estimate and null hypothesis has weights of 1/3 for all condition means. The weights from all three model parameters / hypotheses that were defined are now combined and written into a matrix that we refer to as the *hypothesis matrix* (`Hc`):

```
HcSum <- rbind(cH00=c(low= 1/3, med= 1/3, hi= 1/3),
               cH01=c(low=+2/3, med=-1/3, hi=-1/3),
               cH02=c(low=-1/3, med=+2/3, hi=-1/3))
fractions(t(HcSum))
```

```
##      cH00 cH01 cH02
## low   1/3  2/3 -1/3
## med   1/3 -1/3  2/3
## hi    1/3 -1/3 -1/3
```

Each set of weights is first entered as a row into the matrix (command `rbind()`). This has mathematical reasons (see Schad et al., 2020b). However, we then switch rows and columns of the matrix for easier readability using the command `t()` (this transposes the matrix, i.e., switches rows and columns). The command `fractions()` turns the decimals into fractions to improve readability.

Now that the condition weights have been written into the hypothesis matrix, the third step of the procedure is implemented: a matrix operation called the 'generalized matrix inverse'[1] is used to obtain the contrast matrix that is needed to test these hypotheses in a linear model. In R this next step is done using the function `ginv()` from the `MASS` package. Here, we define a function `ginv2()` for nicer formatting of the output.[3]

```
ginv2 <- function(x) # define a function to make the output nicer
  fractions(provideDimnames(ginv(x),base=dimnames(x)[2:1]))
```

Applying the generalized inverse to the hypothesis matrix results in the new matrix `XcSum`. This is the contrast matrix X_c that estimates exactly those comparisons and tests exactly those hypotheses that were specified earlier:

```
(XcSum <- ginv2(HcSum))
```

```
##      cH00 cH01 cH02
## low   1     1    0
## med   1     0    1
## hi    1    -1   -1
```

This contrast matrix corresponds exactly to the sum contrasts described above. In the case of the sum contrast, the contrast matrix looks very different from the hypothesis matrix. The contrast matrix in sum contrasts codes with +1 the condition that is to be compared to the GM. The condition that is never compared to the GM is coded as −1. Unless we know the relationship between

[1] At this point, there is no need to understand in detail what this means. We refer the interested reader to Schad et al. (2020b). For a quick overview, we recommend a vignette explaining the generalized inverse in the matlib package[2] (Friendly et al., 2020).

[3] The function `fractions()` from the `MASS` package is used to make the output more easily readable, and the function `provideDimnames()` is used to keep row and column names.

the hypothesis matrix and the contrast matrix, the meaning of the coefficients is completely opaque.

To verify this custom-made contrast matrix, it is compared to the sum contrast matrix as generated by the R function `contr.sum()` in the `stats` package. The resulting contrast matrix is identical to the result when adding the intercept term, a column of ones, to the contrast matrix:

```
fractions(cbind(1,contr.sum(3)))
```

```
##   [,1] [,2] [,3]
## 1    1    1    0
## 2    1    0    1
## 3    1   -1   -1
```

In order to estimate model parameters, step four in our procedure involves assigning sum contrasts to the factor `F` in our example data, and running a linear (mixed) model. This allows us to estimate the regression coefficients associated with each contrast. We compare these to the data shown above (Table 6.2) to test whether the regression coefficients actually correspond to the differences of condition means, as intended. To define the contrast, it is necessary to remove the intercept term, as this is automatically added by the linear model function `lm()`.

```
contrasts(df_contrasts2$F) <- XcSum[,2:3]
fit_Sum <- lm(DV ~ 1 + F,
                data = df_contrasts2)
```

```
round(summary(fit_Sum)$coefficients,1)
```

```
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)    450.2        5.8    78.1        0
## FcH01          -49.9        8.2    -6.1        0
## FcH02           49.8        8.2     6.1        0
```

The linear model regression coefficients show the GM response time of 450 ms in the intercept. Remember that the first regression coefficient `FcH01` was designed to estimate in how far nouns are responded to slower than the GM. The regression coefficient `FcH01` ('Estimate') of 50 reflects the difference between nouns (500 ms) and the GM of 450 ms. The estimate of interest was in how response times for verbs differ from the GM. The fact that the second regression coefficient `FcH02` is close to 0 indicates that response times for verbs (450 ms) are the same as the GM of 450 ms. While the nouns are estimated to have 50 ms longer reading times than the GM, the difference in reading times between verbs and GM is estimated to be 0.

We have now not only derived contrasts, parameter estimates, and hypotheses for the sum contrast, we have also used a powerful and highly general procedure that is used to generate contrasts for many kinds of different hypotheses and experimental designs.

6.2.3 Generating contrasts: The `hypr` package

To work with the 4-step procedure, i.e., to flexibly design contrasts to estimate specific comparisons, we have developed the R package `hypr` (Rabe et al., 2020). This package allows the user to specify comparisons as null hypotheses, and based on these null hypotheses, it automatically generates contrast matrices that allow the user to estimate these comparisons and test these hypotheses in linear models. It thus considerably simplifies the implementation of the 4-step procedure outlined above.

To illustrate the functionality of the `hypr` package, we will use the two comparisons and associated null hypotheses that we had defined and analyzed in the previous section:

$$\beta_1 = \mu_1 - \frac{\mu_1 + \mu_2 + \mu_3}{3} = \mu_1 - GM \tag{6.33}$$

$$H_{0_1} : \mu_1 = \frac{\mu_1 + \mu_2 + \mu_3}{3} = GM \tag{6.34}$$

and

$$\beta_2 = \mu_2 - \frac{\mu_1 + \mu_2 + \mu_3}{3} = \mu_2 - GM \tag{6.35}$$

$$H_{0_2} : \mu_2 = \frac{\mu_1 + \mu_2 + \mu_3}{3} = GM \tag{6.36}$$

These null hypotheses are effectively comparisons between condition means or between bundles of condition means. That is, μ_1 is compared to the GM and μ_2 is compared to the grand mean. These two comparisons/hypotheses can be directly entered into R using the `hypr()` function from the `hypr` package. To do so, we use some labels to indicate factor levels. E.g., `nouns`, `verbs`, and `adjectives` can represent factor levels μ_1, μ_2, and μ_3. The first comparison/hypothesis specifies that $\mu_1 = \frac{\mu_1+\mu_2+\mu_3}{3}$. This can be written as a formula in R: `low ~ (low + medium + high)/3`. The second comparison/hypothesis is that $\mu_2 = \frac{\mu_1+\mu_2+\mu_3}{3}$, which can be written in R as `medium ~ (low + medium + high)/3`.

```
HcSum <- hypr(b1 = nouns ~ (nouns + verbs + adjectives)/3,
              b2 = verbs ~ (nouns + verbs + adjectives)/3,
              levels=c("nouns","verbs","adjectives"))
HcSum
```

```
## hypr object containing 2 null hypotheses:
## H0.b1: 0 = 2/3*nouns - 1/3*verbs - 1/3*adjectives
## H0.b2: 0 = 2/3*verbs - 1/3*nouns - 1/3*adjectives
##
## Hypothesis matrix (transposed):
##             b1   b2
## nouns        2/3 -1/3
## verbs       -1/3  2/3
## adjectives -1/3 -1/3
##
## Contrast matrix:
##             b1 b2
```

```
## nouns       1  0
## verbs       0  1
## adjectives -1 -1
```

The results show that the null hypotheses or comparisons between condition means have been re-written into a form, where 0 is coded on the left side of the equation, and the condition means together with associated weights are written on the right side of the equation. This presentation makes it easy to see the weights of the condition means to code a certain null hypothesis or comparison. The next part of the results shows the hypothesis matrix, which contains the weights from the condition means. Thus, `hypr` takes as input comparisons between condition means, which also define null hypotheses, and automatically extracts the corresponding weights and encodes them into the hypothesis matrix. `hypr` moreover applies the generalized matrix inverse to obtain the contrast matrix from the hypothesis matrix. Note that the different steps correspond exactly to the steps we had carried out manually in the preceeding section. `hypr` automatically performs these steps for us. We can now extract the contrast matrix by a simple function call:

```
contr.hypothesis(HcSum)
```

```
##            b1 b2
## nouns       1  0
## verbs       0  1
## adjectives -1 -1
```

We can assign this contrast to our factor as we did before, and again fit a linear model:

```
contrasts(df_contrasts2$F) <- contr.hypothesis(HcSum)
fit_Sum2 <- lm(DV ~ 1 + F,
                data = df_contrasts2)
```

```
round(summary(fit_Sum2)$coefficients,1)
```

```
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)    450.2        5.8    78.1        0
## Fb1            -49.9        8.2    -6.1        0
## Fb2             49.8        8.2     6.1        0
```

In the `hypr`, the focus lies on estimation of contrasts that code comparisons between condition means or groups of condition means. Thus, the null hypotheses that one specifies implicitly imply the estimation of a difference between condition means or bundles of condition means. The output of the `hypr()` function (see the first section of the results) makes this clear - these formulate the null hypotheses in a way that also illustrates the estimation of model parameters. I.e., the null hypothesis `H0.b1: 0 = 2/3*m1 - 1/3*m2 - 1/3*m3` corresponds to a parameter estimate of `b1 = 2/3*m1 - 1/3*m2 - 1/3*m3`. The resulting contrasts will then allow us to estimate the specified differences between condition means or bundles of condition means.

6.3 Further examples of contrasts illustrated with a factor with four levels

In order to understand repeated difference and polynomial contrasts, it may be instructive to consider an experiment with one between-subject factor with four levels. We load a corresponding data set, which contains simulated data about response times with a four-level between-subject factor. The sample sizes for each level and the means and standard errors are shown in Table 6.3, and the means and standard errors are also shown graphically in Figure 6.2.

```
## [1] 20
```

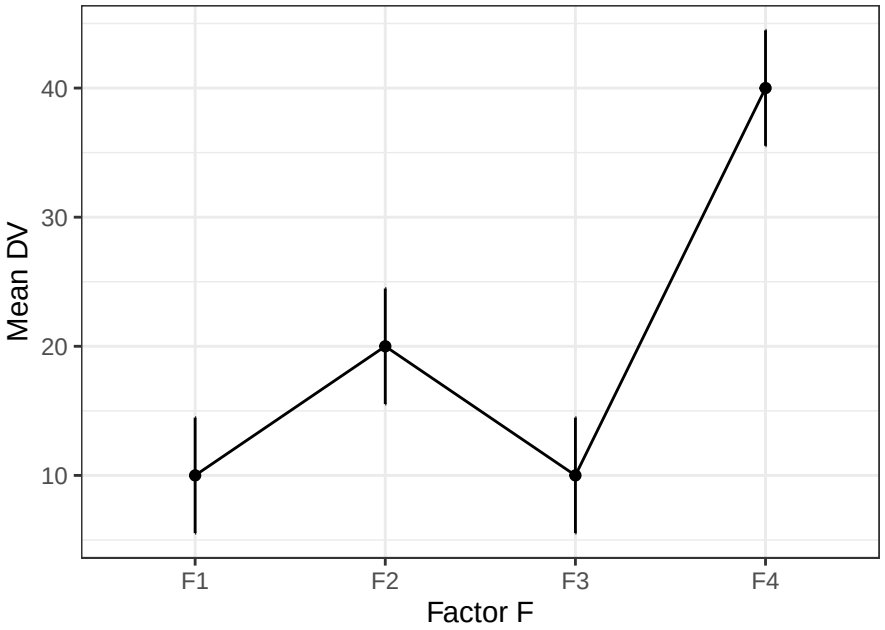


FIGURE 6.2: Means and error bars (showing standard errors) for a simulated data-set with one between-subjects factor with four levels.

```
kbl(table3a, position="b", digits=1,
    caption="Summary statistics per condition for the simulated data.")
```

We assume that the four factor levels `F1` to `F4` reflect levels of word frequency, including the levels `low`, `medium-low`, `medium-high`,

TABLE 6.3: Summary statistics per condition for the simulated data.

Factor	N data	Est. means	Std. dev.	Std. errors
F1	5	10	10	4.5
F2	5	20	10	4.5
F3	5	10	10	4.5
F4	5	40	10	4.5

and `high` frequency words, and that the dependent variable reflects some response time.[4]

6.3.1 Repeated contrasts

A popular contrast psychologists and psycholinguists are interested in is the comparison between neighboring levels of a factor. This type of contrast is called repeated contrast. In our example, our research question might be whether the frequency level `low` leads to slower response times than frequency level `medium-low`, whether frequency level `medium-low` leads to slower response times than frequency level `medium-high`, and whether frequency level `medium-high` leads to slower response times than frequency level `high`.

Repeated contrasts are used to implement these comparisons. Consider first how to derive the contrast matrix for repeated contrasts, starting out by specifying the hypotheses that are to be tested about the data. Importantly, this again applies the general strategy of how to translate (any) hypotheses about differences between groups or conditions into a set of contrasts, yielding a powerful tool of great value in many research settings. We follow the four-step procedure outlined above.

The first step is to specify our comparisons or hypotheses, and to write them down in a way such that their weights can be extracted easily. For a four-level factor, the three hypotheses are:

$$H_{0_{2-1}} : -1 \cdot \mu_1 + 1 \cdot \mu_2 + 0 \cdot \mu_3 + 0 \cdot \mu_4 = 0 \tag{6.37}$$

$$H_{0_{3-2}} : 0 \cdot \mu_1 - 1 \cdot \mu_2 + 1 \cdot \mu_3 + 0 \cdot \mu_4 = 0 \tag{6.38}$$

$$H_{0_{4-3}} : 0 \cdot \mu_1 + 0 \cdot \mu_2 - 1 \cdot \mu_3 + 1 \cdot \mu_4 = 0 \tag{6.39}$$

[4] Qualitatively, the simulated pattern of results is quite similar to the empirically observed values for word frequency effects on single fixation durations (Heister et al., 2012).

Here, the μ_x are the mean response times in condition x. Each hypothesis gives weights to the different condition means. The first hypothesis ($H_{0_{2-1}}$) tests the difference between condition mean for `F2` (μ_2) minus the condition mean for `F1` (μ_1), but ignores condition means for `F3` and `F4` (μ_3, μ_4). μ_1 has a weight of -1, μ_2 has a weight of $+1$, and μ_3 and μ_4 have weights of 0. As the second step, the vector of weights for the first hypothesis is extracted as `c2vs1 <- c(F1=-1,F2=+1,F3=0,F4=0)`. Next, the same thing is done for the other hypotheses - the weights for all hypotheses are extracted and coded into a *hypothesis matrix* in R:

```
t(HcRE <- rbind(c2vs1=c(F1=-1,F2=+1,F3= 0,F4= 0),
                c3vs2=c(F1= 0,F2=-1,F3=+1,F4= 0),
                c4vs3=c(F1= 0,F2= 0,F3=-1,F4=+1)))
```

```
##    c2vs1 c3vs2 c4vs3
## F1    -1     0     0
## F2     1    -1     0
## F3     0     1    -1
## F4     0     0     1
```

Again, we show the transposed version of the hypothesis matrix (switching rows and columns), but now we leave out the hypothesis for the intercept (below, we discuss when this can be ignored).

Next, the new contrast matrix `XcRE` is obtained. This is the contrast matrix X_c that exactly tests the hypotheses written down above:

```
(XcRE <- ginv2(HcRE))
```

```
##    c2vs1 c3vs2 c4vs3
## F1 -3/4  -1/2  -1/4
## F2  1/4  -1/2  -1/4
## F3  1/4   1/2  -1/4
## F4  1/4   1/2   3/4
```

In the case of the repeated contrast, the contrast matrix again

looks very different from the hypothesis matrix. In this case, the contrast matrix looks a lot less intuitive than the hypothesis matrix, and if one did not know the associated hypothesis matrix, it seems unclear what the contrast matrix would actually test. To verify this custom-made contrast matrix, we compare it to the repeated contrast matrix as generated by the R function `contr.sdif()` in the `MASS` package (Ripley, 2019). The resulting contrast matrix is identical to our result:

```
fractions(contr.sdif(4))
```

```
##   2-1  3-2  4-3
## 1 -3/4 -1/2 -1/4
## 2  1/4 -1/2 -1/4
## 3  1/4  1/2 -1/4
## 4  1/4  1/2  3/4
```

Step four in the procedure is to apply repeated contrasts to the factor `F` in the example data, and to run a linear model. This allows us to estimate the regression coefficients associated with each contrast. These are compared to the data in Figure 6.2 to test whether the regression coefficients actually correspond to the differences between successive condition means, as intended.

```
contrasts(df_contrasts3$F) <- XcRE
fit_Rep <- lm(DV ~ 1 + F,
                 data = df_contrasts3)
```

```
round(summary(fit_Rep)$coefficients,1)
```

```
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)       20        2.2     8.9      0.0
## Fc2vs1            10        6.3     1.6      0.1
## Fc3vs2           -10        6.3    -1.6      0.1
## Fc4vs3            30        6.3     4.7      0.0
```

The results show that, as expected, the regression coefficients reflect the differences that were of interest: the regression coefficient ('Estimate') `Fc2vs1` has a value of 10, which exactly corresponds to the difference between the condition mean for `F2` (20) minus the condition mean for `F1` (10), i.e., $20-10=10$. Likewise, the regression coefficient `Fc3vs2` has a value of -10, which corresponds to the difference between the condition mean for `F3` (10) minus the condition mean for `F2` (20), i.e., $10-20=-10$. Finally, the regression coefficient `Fc4vs3` has a value of roughly 30, which reflects the difference between condition `F4` (40) minus condition `F3` (10), i.e., $40-10=30$. Thus, the regression coefficients estimate differences between successive or neighboring condition means, and test the corresponding null hypotheses.

Again, we can perform the same computations automatically using the `hypr` package. For reasons of brevity, we here only show the relevant `hypr` command: `hypr(b1=F1~F2, b2=F2~F3, b3=F3~F4)`, and leave it to the reader to investigate this more closely.

6.3.2 Contrasts in linear regression analysis: The design or model matrix

We have now discussed how different contrasts are created from the hypothesis matrix. However, we have not treated in detail how exactly contrasts are used in a linear model. Here, we will see that the contrasts for a factor in a linear model are just the same thing as continuous numeric predictors (i.e., covariates) in a linear/multiple regression analysis. That is, contrasts are the way to encode discrete factor levels into numeric predictor variables to use in linear/multiple regression analysis, by encoding which differences between factor levels are tested. The contrast matrix X_c that we have looked at so far has one entry (row) for each experimental condition. For use in a linear model, however, the contrast matrix is coded into a design or model matrix X, where each individual data point has one row. The design matrix X can be extracted using the function `model.matrix()`:

```
(contrasts(df_contrasts3$F) <- XcRE) # contrast matrix
```

```
##    c2vs1 c3vs2 c4vs3
## F1 -3/4  -1/2  -1/4
## F2  1/4  -1/2  -1/4
## F3  1/4   1/2  -1/4
## F4  1/4   1/2   3/4
```

```
covars <- model.matrix(~ 1 + F, df_contrasts3) # design matrix
(covars <- as.data.frame(covars))
```

```
##    (Intercept) Fc2vs1 Fc3vs2 Fc4vs3
## 1            1  -0.75   -0.5  -0.25
## 2            1  -0.75   -0.5  -0.25
## 3            1  -0.75   -0.5  -0.25
## 4            1  -0.75   -0.5  -0.25
## 5            1  -0.75   -0.5  -0.25
## 6            1   0.25   -0.5  -0.25
## 7            1   0.25   -0.5  -0.25
## 8            1   0.25   -0.5  -0.25
## 9            1   0.25   -0.5  -0.25
## 10           1   0.25   -0.5  -0.25
## 11           1   0.25    0.5  -0.25
## 12           1   0.25    0.5  -0.25
## 13           1   0.25    0.5  -0.25
## 14           1   0.25    0.5  -0.25
## 15           1   0.25    0.5  -0.25
## 16           1   0.25    0.5   0.75
## 17           1   0.25    0.5   0.75
## 18           1   0.25    0.5   0.75
## 19           1   0.25    0.5   0.75
## 20           1   0.25    0.5   0.75
```

For each of the 20 subjects, four numbers are stored in this model matrix. They represent the three values of three predictor variables used to predict response times in the task. Indeed, this matrix is

exactly the design matrix X commonly used in multiple regression analysis, where each column represents one numeric predictor variable (covariate), and the first column codes the intercept term.

To further illustrate this, the covariates are extracted from this design matrix and stored separately as numeric predictor variables in the data-frame:

```
df_contrasts3[,c("Fc2vs1","Fc3vs2","Fc4vs3")] <- covars[,2:4]
```

They are now used as numeric predictor variables in a multiple regression analysis:

```
fit_m3 <- lm(DV ~ 1 + Fc2vs1 + Fc3vs2 + Fc4vs3,
                 data = df_contrasts3)
```

```
round(summary(fit_m3)$coefficients,1)
```

```
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)       20        2.2     8.9      0.0
## Fc2vs1            10        6.3     1.6      0.1
## Fc3vs2           -10        6.3    -1.6      0.1
## Fc4vs3            30        6.3     4.7      0.0
```

The results show that the regression coefficients are the same as in the contrast-based analysis shown in the previous section. This demonstrates that contrasts serve to code discrete factor levels into a linear/multiple regression analysis by numerically encoding comparisons between specific condition means.

6.3.3 Polynomial contrasts

Polynomial contrasts are another option for analyzing factors. Suppose that we expect a linear trend across conditions, where the response increases by a constant magnitude with each successive factor level. This could be the expectation when four levels of a factor reflect decreasing levels of word frequency (i.e., four factor

levels: high, medium-high, medium-low, and low word frequency), where one expects the lowest response for high frequency words, and successively higher responses for lower word frequencies. The effect for each individual level of a factor may not be strong enough for detecting it in the statistical model. Specifying a linear trend in a polynomial constrast allows us to pool the whole increase into a single coefficient for the linear trend, increasing statistical power to detect the increase. Such a specification constrains the estimate to one interpretable parameter, e.g., a linear increase across factor levels. The larger the number of factor levels, the more parsimonious are polynomial contrasts compared to contrast-based specifications as introduced in the previous sections. Going beyond a linear trend, one may also have expectations about quadratic trends. For example, one may expect an increase only among very low frequency words, but no difference between high and medium-high frequency words.

```
Xpol <- contr.poly(4)
(contrasts(df_contrasts3$F) <- Xpol)
```

```
##            .L   .Q      .C
## [1,] -0.6708  0.5 -0.2236
## [2,] -0.2236 -0.5  0.6708
## [3,]  0.2236 -0.5 -0.6708
## [4,]  0.6708  0.5  0.2236
```

```
fit_Pol <- lm(DV ~ 1 + F,
                data = df_contrasts3)
```

```
round(summary(fit_Pol)$coefficients,1)
```

```
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)     20.0        2.2     8.9        0
## F.L             17.9        4.5     4.0        0
## F.Q             10.0        4.5     2.2        0
```

```
## F.C              13.4        4.5      3.0         0
```

In this example, condition means increase across factor levels in a linear fashion, but there may also be quadratic and cubic trends.

6.4 What makes a good set of contrasts?

For a factor with I levels one can make $I - 1$ comparisons. For example, in a design with one factor with two levels, only one comparison is possible (between the two factor levels). More generally, if we have a factor with I_1 and another factor with I_2 levels, then the total number of conditions is $I_1 \times I_2 = \nu$ (not $I_1 + I_2$!), which implies a maximum of $\nu - 1$ contrasts.

For example, in a design with one factor with three levels, A, B, and C, in principle one could make three comparisons (A vs. B, A vs. C, B vs. C). However, after defining an intercept, only two means can be compared. Therefore, for a factor with three levels, we define two comparisons within one statistical model. F tests are nothing but combinations, or bundles of contrasts. F tests are less specific and they lack focus, but they are useful when the hypothesis in question is vague. However, a significant F test leaves unclear what effects the data actually show. Contrasts are very useful to test specific effects in the data.

One critical precondition for contrasts is that they implement different hypotheses that are not collinear, that is, that none of the contrasts can be generated from the other contrasts by linear combination. For example, the contrast `c1 = c(1,2,3)` can be generated from the contrast `c2 = c(3,4,5)` simply by computing `c2 - 2`. Therefore, contrasts c1 and c2 cannot be used simultaneously. That is, each contrast needs to encode some independent information about the data.

There are (at least) two criteria to decide what a good contrast is. First, *orthogonal contrasts* have advantages as they test mutually independent hypotheses about the data (see Dobson and Barnett,

2011, section 6.2.5, p. 91 for a detailed explanation of orthogonality). Second, it is crucial that contrasts are defined in a way such that they answer the research questions. This second point is crucial. One way to accomplish this, is to use the hypothesis matrix to generate contrasts (e.g., via the `hypr` package), as this ensures that one uses contrasts that exactly estimate the comparisons of interest in a given study.

6.4.1 Centered contrasts

Contrasts are often constrained to be centered, such that the individual contrast coefficients c_i for different factor levels i sum to 0: $\sum_{i=1}^{I} c_i = 0$. This has advantages when testing interactions with other factors or covariates (we discuss interactions between factors in a separate chapter below). All contrasts discussed here are centered except for the treatment contrast, in which the contrast coefficients for each contrast do not sum to zero:

```
colSums(contr.treatment(4))
```

```
## 2 3 4
## 1 1 1
```

Other contrasts, such as repeated contrasts, are centered and the contrast coefficients for each contrast sum to 0:

```
colSums(contr.sdif(4))
```

```
## 2-1 3-2 4-3
##   0   0   0
```

The contrast coefficients mentioned above appear in the contrast matrix. By contrast, the weights in the hypothesis matrix are always centered. This is also true for the treatment contrast. The reason is that they code hypotheses, which always relate to comparisons between conditions or bundles of conditions. The only exception are the weights for the intercept, which are all the same and together always sum to 1 in the hypothesis matrix. This is

done to ensure that when applying the generalized matrix inverse, the intercept results in a constant term with values of 1 in the contrast matrix. An important question concerns whether (or when) the intercept needs to be considered in the generalized matrix inversion, and whether (or when) it can be ignored. This question is closely related to the concept of orthogonal contrasts, a concept we turn to below.

6.4.2 Orthogonal contrasts

Two centered contrasts c_1 and c_2 are orthogonal to each other if the following condition applies. Here, i is the i-th cell of the vector representing the contrast.

$$\sum_{i=1}^{I} c_{1,i} \cdot c_{2,i} = 0 \tag{6.40}$$

Orthogonality can be determined easily in R by computing the correlation between two contrasts. Orthogonal contrasts have a correlation of 0. Contrasts are therefore just a special case for the general case of predictors in regression models, where two numeric predictor variables are orthogonal if they are un-correlated.

For example, coding two factors in a 2×2 design (we return to this case in a section on designs with two factors below) using sum contrasts, these sum contrasts and their interaction are orthogonal to each other:

```
(Xsum <- cbind(F1=c(1,1,-1,-1), F2=c(1,-1,1,-1), F1xF2=c(1,-1,-1,1)))
```

```
##      F1 F2 F1xF2
## [1,]  1  1     1
## [2,]  1 -1    -1
## [3,] -1  1    -1
## [4,] -1 -1     1
```

```
cor(Xsum)
```

```
##       F1 F2 F1xF2
## F1     1  0     0
## F2     0  1     0
## F1xF2  0  0     1
```

Notice that the correlations between the different contrasts (i.e., the off-diagonals) are exactly 0. Sum contrasts coding one multi-level factor, however, are not orthogonal to each other:

```
cor(contr.sum(4))
```

```
##      [,1] [,2] [,3]
## [1,]  1.0  0.5  0.5
## [2,]  0.5  1.0  0.5
## [3,]  0.5  0.5  1.0
```

Here, the correlations between individual contrasts, which appear in the off-diagonals, deviate from 0, indicating non-orthogonality. The same is also true for treatment and repeated contrasts:

```
cor(contr.sdif(4))
```

```
##        2-1    3-2    4-3
## 2-1 1.0000 0.5774 0.3333
## 3-2 0.5774 1.0000 0.5774
## 4-3 0.3333 0.5774 1.0000
```

```
cor(contr.treatment(4))
```

```
##         2       3       4
## 2  1.0000 -0.3333 -0.3333
## 3 -0.3333  1.0000 -0.3333
## 4 -0.3333 -0.3333  1.0000
```

Orthogonality of contrasts plays a critical role when computing the generalized inverse. In the inversion operation, orthogonal contrasts are converted independently from each other. That is, the presence or absence of another orthogonal contrast does not change the resulting weights. In fact, for orthogonal contrasts, applying the generalized matrix inverse to the hypothesis matrix simply produces a scaled version of the hypothesis matrix into the contrast matrix (for mathematical details, see Schad et al. (2020b)).

6.4.3 The role of the intercept in non-centered contrasts

A related question concerns whether the intercept needs to be considered when computing the generalized inverse for a contrast. It turns out that considering the intercept is necessary for contrasts that are not centered. This is the case for treatment contrasts which are not centered; e.g., the treatment contrast for two factor levels `c1vs0 = c(0,1)`: $\sum_i c_i = 0 + 1 = 1$. One can actually show that the formula to determine whether contrasts are centered (i.e., $\sum_i c_i = 0$) is the same formula as the formula to test whether a contrast is "orthogonal to the intercept". Remember that for the intercept, all contrast coefficients are equal to one: $c_{1,i} = 1$ (here, $c_{1,i}$ indicates the vector of contrast coefficients associated with the intercept). We enter these contrast coefficient values into the formula testing whether a contrast is orthogonal to the intercept (here, $c_{2,i}$ indicates the vector of contrast coefficients associated with some contrast for which we want to test whether it is "orthogonal to the intercept"): $\sum_i c_{1,i} \cdot c_{2,i} = \sum_i 1 \cdot c_{2,i} = \sum_i c_{2,i} = 0$. The resulting formula is: $\sum_i c_{2,i} = 0$, which is exactly the formula for whether a contrast is centered. Because of this analogy, treatment contrasts can be viewed to be 'not orthogonal to the intercept'. This means that the intercept needs to be considered when computing the generalized inverse for treatment contrasts. As we have discussed above, when the intercept is included in the hypothesis matrix, the weights for this intercept term should sum to one, as this yields a column of ones for the intercept term in the contrast matrix.

We can see that considering the intercept makes a difference for the

treatment contrast. First, we define the comparisons involved in a treatment contrast, where two experimental conditions `m1` and `m2` are each compared to a baseline condition `m0` (`m0~m1` and `m0~m2`). In addition, we explicitly code the intercept term, which involves a comparison of the baseline to 0 (`m0~0`). We take a look at the resulting contrast matrix:

```
hypr(int=m0~0, b1=m1~m0, b2=m2~m0)
```

```
## hypr object containing 3 null hypotheses:
## H0.int: 0 = m0
##  H0.b1: 0 = m1 - m0
##  H0.b2: 0 = m2 - m0
##
## Hypothesis matrix (transposed):
##    int b1 b2
## m0   1  -1 -1
## m1   0   1  0
## m2   0   0  1
##
## Contrast matrix:
##    int b1 b2
## m0 1   0  0
## m1 1   1  0
## m2 1   0  1
```

```
contr.treatment(c("m0","m1","m2"))
```

```
##    m1 m2
## m0  0  0
## m1  1  0
## m2  0  1
```

This shows a contrast matrix that we know from the treatment contrast. The intercept is coded as a column of 1s. And each of the comparisons is coded as a 1 in the condition which is compared to the baseline, and a 0 in other conditions. The point is here that

this gives us the contrast matrix that is expected and known for the treatment contrast.

However, we can also ignore the intercept in the specification of the hypotheses:

```
hypr(b1=m1~m0, b2=m2~m0)
```

```
## hypr object containing 2 null hypotheses:
## H0.b1: 0 = m1 - m0
## H0.b2: 0 = m2 - m0
##
## Hypothesis matrix (transposed):
##    b1 b2
## m0 -1 -1
## m1  1  0
## m2  0  1
##
## Contrast matrix:
##    b1   b2
## m0 -1/3 -1/3
## m1  2/3 -1/3
## m2 -1/3  2/3
```

Interestingly, the resulting contrast matrix now looks very different from the contrast matrix that we know from the treatment contrast. Indeed, this contrast also estimates a reasonable set of quantities. It again estimates how strongly condition mean `m1` differs from the baseline and how `m2` differs from baseline. The intercept, however, now estimates the average dependent variable across all three conditions (i.e., the GM). This can be seen by explicitly adding a comparison of the average of all three conditions to 0:

```
hypr(int=(m0+m1+m2)/3~0, b1=m1~m0, b2=m2~m0)
```

```
## hypr object containing 3 null hypotheses:
```

```
## H0.int: 0 = 1/3*m0 + 1/3*m1 + 1/3*m2
##  H0.b1: 0 = m1 - m0
##  H0.b2: 0 = m2 - m0
##
## Hypothesis matrix (transposed):
##    int b1  b2
## m0 1/3  -1  -1
## m1 1/3   1   0
## m2 1/3   0   1
##
## Contrast matrix:
##    int  b1   b2
## m0    1 -1/3 -1/3
## m1    1  2/3 -1/3
## m2    1 -1/3  2/3
```

The resulting contrast matrix is now the same as when the intercept was ignored, which confirms that these both test the same hypotheses.

6.5 Summary

To summarize, contrasts provide a way to tell the linear (mixed-effects) model how to code factors into numeric covariates. That is, they provide a way to define which comparisons between which condition means or bundles of condition means should be estimated in the Bayesian model. There are a number of default contrasts, like treatment contrasts, sum contrasts, repeated contrasts, or Helmert contrasts, that are known to test specific hypotheses about the data. A much more powerful procedure is to use the generalized matrix inverse, e.g., as implemented in the `hypr` package, to derive contrasts automatically after specifying the comparisons that a contrast should estimate.

```
library(knitr)
library(kableExtra)
library(tidyverse)
library(brms)
```

```
## Loading required package: Rcpp
```

```
## Loading 'brms' package (version 2.14.4). Useful instructions
## can be found by typing help('brms'). A more detailed introduction
## to the package is available through vignette('brms_overview').
```

```
##
## Attaching package: 'brms'
```

```
## The following object is masked from 'package:lme4':
##
##     ngrps
```

```
## The following objects are masked from 'package:extraDistr':
##
##     ddirichlet, dfrechet, pfrechet, qfrechet,
##     rdirichlet, rfrechet
```

```
## The following object is masked from 'package:stats':
##
##     ar
```

```
library(bcogsci)
library(papaja)
```

```
## Loading required package: tinylabels
```

```
##
## Attaching package: 'papaja'
```

```
## The following object is masked _by_ '.GlobalEnv':
##
##     ci
```

```
library(afex)
```

```
## ************
## Welcome to afex. For support visit: http://afex.singmann.science/
## - Functions for ANOVAs: aov_car(), aov_ez(), and aov_4()
## - Methods for calculating p-values with mixed(): 'KR', 'S', 'LRT', and 'PB'
## - 'afex_aov' and 'mixed' objects can be passed to emmeans() for follow-up tests
## - NEWS: library('emmeans') now needs to be called explicitly!
## - Get and set global package options with: afex_options()
## - Set orthogonal sum-to-zero contrasts globally: set_sum_contrasts()
## - For example analyses see: browseVignettes("afex")
## ************
##
## Attaching package: 'afex'
## The following object is masked from 'package:lme4':
##
##     lmer
```

```
library(MASS)
library(hypr)
library(bcogsci)
```

7

Contrast coding for designs with two predictor variables

Chapter 6 provides a basic introduction into contrast coding in situations where there is one predictor variable, i.e., one factor, which can be tested using one specified contrast matrix. Here, we will investigate how contrast coding generalizes to situations where there is more than one predictor variables. This could either be a situation where two factors are present or where one factor is paired with a continuous predictor variables, i.e., a covariate. We first discuss contrast coding for the case of two factors (for 2×2 designs; see section 7.1) and then go on to investigate situations where one predictor is a factor and the other predictor is a continuous covariate (see section 7.2). Moreover, one problem in the analysis of interactions occurs in situations where the model is not linear, but has some non-linear link function, such as e.g., in logistic models or when assuming a log-normally distributed dependent variable. In these situations, the model makes predictions for each condition (i.e., design cell) at the latent level of the linear model. However, sometimes it is important to translate these model predictions to the level of the observations (e.g., to probabilities in a logistic regression model). We will discuss how this can be implemented in section 7.3. Now, we first start by treating contrast coding in a factorial 2 x 2 design.

7.1 Contrast coding in a factorial 2 x 2 design

In chapter 6 in section 6.3, we have used a data set with one 4-level factor. Here, we assume that the exact same four means

come from an $A(2) \times B(2)$ between-subject-factor design rather than an F(4) between-subject-factor design. We load the artificial, simulated data and show summary statistics in Table 7.1 and in Figure 7.1. The means and standard deviations are exactly the same as in Figure 6.2 and in Table 6.3.

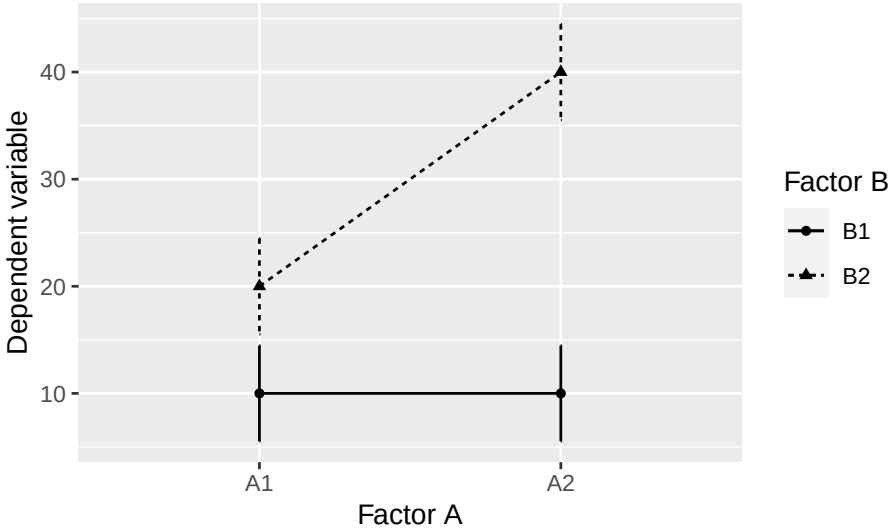


FIGURE 7.1: Means and error bars (showing standard errors) for a simulated data-set with a two-by-two between-subjects factorial design.

TABLE 7.1: Summary statistics per condition for the simulated data.

Factor A	Factor B	N data	Means	Std. dev.	Std. errors
A1	B1	5	10	10	4.5
A1	B2	5	20	10	4.5
A2	B1	5	10	10	4.5
A2	B2	5	40	10	4.5

7.1.1 The difference between an ANOVA and a multiple regression

Let's compare the traditional ANOVA with multiple regression for analyzing these data.

```
# ANOVA: B_A(2) times B_B(2)
fit_AB_aov<-aov_car(DV ~ A*B + Error(id), data=df_contrasts4)

# MR: B_A(2) times B_B(2)
fit_AB_mr <- lm(DV ~ 1 + A*B, data=df_contrasts4)
```

The results from the two analyses, shown in the R output and in Table **??**, are very different. How do we see these are different? Notice that it is possible to compute F-values from t-values from the fact that $F(1, df) = t(df)^2$ (Snedecor and Cochran, 1967) (where df indicates degrees of freedom). When applying this to the above multiple regression model, the F-value for factor A (i.e., $AA2$) is $0.00^2 = 0$. This is obviously not the same as in the ANOVA, where the F-value for factor A is 5. Likewise, in the multiple regression factor B (i.e., $BB2$) has an F-value of $1.58^2 = 2.5$, which also does not correspond to the F-value for factor B in the ANOVA of 20. Interestingly, however, the F-value for the interaction is identical in both models, as $2.24^2 = 5$.

The reason that the results from the ANOVA and the results from the multiple regression are different is that one needs sum contrasts in the linear model to get the conventional tests from an ANOVA model. (This is true for factors with two levels, but does not generalize to factors with more levels.)

```
# define sum contrasts:
contrasts(df_contrasts4$A) <- contr.sum(2)
contrasts(df_contrasts4$B) <- contr.sum(2)
# frequentist LM
fit_AB_mr.sum <- lm(DV ~ 1 + A*B, data=df_contrasts4)
```

When using sum contrasts, the results from the multiple regression

TABLE 7.2: Regression analysis with sum contrasts.

Predictor	Estimate	Std. Error	t	p
(Intercept)	20	2.236	8.944	0.00
A1	-5	2.236	-2.236	0.04
B1	-10	2.236	-4.472	0.00
A1:B1	5	2.236	2.236	0.04

models (see Table 7.2) are identical to the results from the ANOVA (see R output). Factor A now has $t^2 = -2.24^2 = 5$, factor B has $t^2 = -4.47^2 = 20$, and the interaction has $t^2 2.24^2 = 5$. All F-values are now the same as in the ANOVA model.

Next, we reproduce the $A(2) \times B(2)$ - ANOVA with contrasts specified for the corresponding one-way $F(4)$ ANOVA, that is by treating the $2 \times 2 = 4$ condition means as four levels of a single factor F. In other words, we go back to the data frame simulated for the analysis of repeated contrasts (see chapter 6, section 6.3). We first define weights for condition means according to our hypotheses, invert this matrix, and use it as the contrast matrix for factor F in an LM. We define weights of $1/4$ and $-1/4$. We do so because (a) we want to compare the mean of two conditions to the mean of two other conditions (e.g., factor A compares $\frac{F1+F2}{2}$ to $\frac{F3+F4}{2}$). Moreover, (b) we want to use sum contrasts, where the regression coefficients assess half the difference between means. Together (a+b), this yields weights of $1/2 \cdot 1/2 = 1/4$. The resulting contrast matrix contains contrast coefficients of $+1$ or -1, showing that we successfully implemented sum contrasts. The results are identical to the previous models.

```
data("df_contrasts3")
t(fractions(HcInt <- rbind(A  =c(F1=1/4,F2= 1/4,F3=-1/4,F4=-1/4),
                           B  =c(F1=1/4,F2=-1/4,F3= 1/4,F4=-1/4),
                           AxB=c(F1=1/4,F2=-1/4,F3=-1/4,F4= 1/4))))
```

```
##     A     B     AxB
```

```
## F1  1/4  1/4  1/4
## F2  1/4 -1/4 -1/4
## F3 -1/4  1/4 -1/4
## F4 -1/4 -1/4  1/4
```

```
(XcInt <- ginv2(HcInt))
```

```
##    A  B  AxB
## F1  1  1  1
## F2  1 -1 -1
## F3 -1  1 -1
## F4 -1 -1  1
```

```
contrasts(df_contrasts3$F) <- XcInt
```

```
fit_F4.sum <- lm(DV ~ 1 + F,
                 data = df_contrasts3)
```

```
round(summary(fit_F4.sum)$coefficients)
```

```
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)       20          2       9        0
## FA                -5          2      -2        0
## FB               -10          2      -4        0
## FAxB               5          2       2        0
```

This shows that it is possible to specify the contrasts not only for each factor (e.g., here in the 2×2 design) separately. One can alternatively pool all experimental conditions (or design cells) into one large factor (here factor F with 4 levels), and specify the contrasts for the main effects and for the interactions in the resulting one large contrast matrix simultaneously.

In this approach, it can again be very useful to apply the `hypr` package to construct contrasts for a 2×2 design. The first hypothesis

estimates the main effect A, i.e., it compares the average of F1 and F2 to the average of F3 and F4. The second parameter estimates the main effect B, i.e., it compares the average of F1 and F3 to the average of F2 and F4. Note that we code direct differences between the averages, i.e., we implement scaled sum contrasts instead of sum contrasts. This becomes clear below as the contrast matrix contains coefficients of $+1/2$ and $-1/2$ instead of $+1$ and -1. The interaction term estimates the difference between differences, i.e., the difference between F1 - F2 and F3 - F4.

```
hAxB <- hypr(A   = (F1+F2)/2~(F3+F4)/2,
             B   = (F1+F3)/2~(F2+F4)/2,
             AxB = (F1-F2)/2~(F3-F4)/2)
hAxB
```

```
## hypr object containing 3 null hypotheses:
##   H0.A: 0 = 1/2*F1 + 1/2*F2 - 1/2*F3 - 1/2*F4
##   H0.B: 0 = 1/2*F1 + 1/2*F3 - 1/2*F2 - 1/2*F4
## H0.AxB: 0 = 1/2*F1 - 1/2*F2 - 1/2*F3 + 1/2*F4
##
## Hypothesis matrix (transposed):
##    A    B    AxB
## F1  1/2  1/2  1/2
## F2  1/2 -1/2 -1/2
## F3 -1/2  1/2 -1/2
## F4 -1/2 -1/2  1/2
##
## Contrast matrix:
##    A    B    AxB
## F1  1/2  1/2  1/2
## F2  1/2 -1/2 -1/2
## F3 -1/2  1/2 -1/2
## F4 -1/2 -1/2  1/2
```

```
contrasts(df_contrasts3$F) <- contr.hypothesis(hAxB)
```

```
fit_F4hypr <- lm(DV ~ 1 + F,
                 data = df_contrasts3)
```

```
round(summary(fit_F4hypr)$coefficients)
```

```
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)       20          2       9        0
## FA               -10          4      -2        0
## FB               -20          4      -4        0
## FAxB              10          4       2        0
```

The results show that the estimates have half the size as compared to the sum contrasts - this is the result of the scaling that we applied. I.e., the main effects now directly estimate the difference between averages. However, both contrasts provide the exact same hypothesis tests. Thus, the hypr package can be used to code hypotheses in a 2 x 2 design.

7.1.2 Nested effects

One can specify hypotheses that do not correspond directly to main effects and interaction of the traditional ANOVA. For example, in a 2×2 experimental design, where factor A codes word frequency (low/high) and factor B is part of speech (noun/verb), one can test the effect of word frequency within nouns and the effect of word frequency within verbs. Formally, A_{B1} versus A_{B2} are nested within levels of B. Differently put, simple effects of factor A are tested for each of the levels of factor B. In this version, we test whether there is a main effect of part of speech (B; as in traditional ANOVA). However, instead of also estimating the second main effect word frequency, A, and the interaction, we estimate (1) whether the two levels of word frequency, A, differ for the first level of B (i.e., nouns) and (2) whether the two levels of word frequency, A, differ for the second level of B (i.e., verbs). In other words, we estimate whether there are differences for A in

each of the levels of B. Often researchers have hypotheses about these differences, and not about the interaction.

```
t(fractions(HcNes <- rbind(B   =c(F1= 1/2,F2=-1/2,F3= 1/2,F4=-1/2),
                           B1xA=c(F1=-1  ,F2= 0  ,F3= 1  ,F4= 0  ),
                           B2xA=c(F1= 0  ,F2=-1  ,F3= 0  ,F4= 1 ))))
```

```
##    B    B1xA B2xA
## F1  1/2   -1    0
## F2 -1/2    0   -1
## F3  1/2    1    0
## F4 -1/2    0    1
```

```
(XcNes <- ginv2(HcNes))
```

```
##    B    B1xA B2xA
## F1  1/2 -1/2    0
## F2 -1/2    0 -1/2
## F3  1/2  1/2    0
## F4 -1/2    0  1/2
```

```
contrasts(df_contrasts3$F) <- XcNes
```

```
fit_Nest <- lm(DV ~ 1 + F,
                 data = df_contrasts3)
```

```
round(summary(fit_Nest)$coefficients)
```

```
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)       20          2       9        0
## FB               -20          4      -4        0
## FB1xA              0          6       0        1
## FB2xA             20          6       3        0
```

Regression coefficients estimate the GM, the difference for the main effect of word frequency (A) and the two differences (for B; i.e., simple main effects) within levels of word frequency (A).

These custom nested contrasts' columns are scaled versions of the corresponding hypothesis matrix. This is the case because the columns are orthogonal. It illustrates the advantage of orthogonal contrasts for the interpretation of regression coefficients: the underlying hypotheses being tested are already clear from the contrast matrix.

There is also a built-in R formula specification of nested designs. The order of factors in the formula from left to right specifies a top-down order of nesting within levels, i.e., here factor A (word frequency) is nested within levels of the factor B (part of speech). This yields the exact same result as our previous result based on custom nested contrasts:

```
contrasts(df_contrasts4$A) <- c(-0.5,+0.5)
contrasts(df_contrasts4$B) <- c(+0.5,-0.5)
fit_Nest2 <- lm(DV ~ 1 + B / A,
                data = df_contrasts4)
```

```
round(summary(fit_Nest2)$coefficients)
```

```
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)       20          2       9        0
## B1               -20          4      -4        0
## BB1:A1             0          6       0        1
## BB2:A1            20          6       3        0
```

xxx

Note that in cases such as these, where A_{B1} vs. A_{B2} are nested within levels of B, it is necessary to include the effect of B (part of speech) in the model, even if one is only interested in the effect of A (word frequency) within levels of B (part of speech). Leaving

out factor B in this case can lead to biases in parameter estimation in the case the data are not fully balanced.

Again, we show how nested contrasts can be easily implemented using `hypr`:

```
hNest <- hypr(B    = (F1+F3)/2~(F2+F4)/2,
              B1xA = F3~F1,
              B2xA = F4~F2)
hNest
```

```
## hypr object containing 3 null hypotheses:
##    H0.B: 0 = 1/2*F1 + 1/2*F3 - 1/2*F2 - 1/2*F4
## H0.B1xA: 0 = F3 - F1
## H0.B2xA: 0 = F4 - F2
##
## Hypothesis matrix (transposed):
##    B    B1xA B2xA
## F1  1/2   -1    0
## F2 -1/2    0   -1
## F3  1/2    1    0
## F4 -1/2    0    1
##
## Contrast matrix:
##    B    B1xA B2xA
## F1  1/2 -1/2    0
## F2 -1/2    0 -1/2
## F3  1/2  1/2    0
## F4 -1/2    0  1/2
```

```
contrasts(df_contrasts3$F) <- contr.hypothesis(hNest)
```

```
fit_NestHypr <- lm(DV ~ 1 + F,
                   data = df_contrasts3)
```

```
round(summary(fit_NestHypr)$coefficients)
```

```
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)       20          2       9        0
## FB               -20          4      -4        0
## FB1xA              0          6       0        1
## FB2xA             20          6       3        0
```

Of course, we can also ask the reverse question: Are there differences for part of speech (B) in the levels of word frequency (A; in addition to estimating the main effect of word frequency, A)? That is, do nouns differ from verbs for low-frequency words (B_{A1}) and do nouns differ from verbs for high-frequency words (B_{A2})?

```
hNest2 <- hypr(A    = (F1+F2)/2~(F3+F4)/2,
               A1xB = F2~F1,
               A2xB = F4~F3)
hNest2
```

```
## hypr object containing 3 null hypotheses:
##    H0.A: 0 = 1/2*F1 + 1/2*F2 - 1/2*F3 - 1/2*F4
## H0.A1xB: 0 = F2 - F1
## H0.A2xB: 0 = F4 - F3
##
## Hypothesis matrix (transposed):
##    A    A1xB A2xB
## F1  1/2   -1    0
## F2  1/2    1    0
## F3 -1/2    0   -1
## F4 -1/2    0    1
##
## Contrast matrix:
##    A    A1xB A2xB
## F1  1/2 -1/2    0
## F2  1/2  1/2    0
## F3 -1/2    0 -1/2
```

```
## F4 -1/2    0  1/2
```

```
contrasts(df_contrasts3$F) <- contr.hypothesis(hNest2)
```

```
fit_Nest2Hypr <- lm(DV ~ 1 + F,
                    data = df_contrasts3)
```

```
round(summary(fit_Nest2Hypr)$coefficients)
```

```
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)       20          2       9        0
## FA               -10          4      -2        0
## FA1xB             10          6       2        0
## FA2xB             30          6       5        0
```

xxx

Regression coefficients estimate the GM, the difference for the main effect of word frequency (A) and the two part of speech effects (for B; i.e., simple main effects) within levels of word frequency (A).

7.1.3 Interactions between contrasts

We have discussed above that in a 2×2 experimental design, the results from sum contrasts are equivalent to typical ANOVA results. In addition, we had also run the analysis with treatment contrasts. It was clear that the results for treatment contrasts (see Table **??**) did not correspond to the results from the ANOVA. However, if the results for treatment contrasts do not correspond to the typical ANOVA results, what do they then test? That is, is it still possible to meaningfully interpret the results from the treatment contrasts in a simple 2×2 design?

This leads us to a very important principle in interpreting results from contrasts: When interactions between contrasts are included in a model, then the results of one contrast actually depend on

the specification of the other contrast(s) in the analysis! This may be counter-intuitive at first. However, it is very important and essential to keep in mind when interpreting results from contrasts. How does this work in detail?

The general rule to remember is that the main effect of one contrast measures its effect at the location 0 of the other contrast(s) in the analysis. What does that mean? Let us consider the example that we use two treatment contrasts in a 2×2 design (see results in Table **??**). Let's take a look at the main effect of factor A. How can we interpret what this measures or tests? This main effect actually tests the effect of factor A at the "location" where factor B is coded as 0. Factor B is coded as a treatment contrast, that is, it codes a zero at its baseline condition, which is B1. Thus, the main effect of factor A tests the effect of A nested within the baseline condition of B. We take a look at the data presented in Figure 7.1, what this nested effect should be. Figure 7.1 shows that the effect of factor A nested in B1 is 0. If we now compare this to the results from the linear model, it is indeed clear that the main effect of factor A (see Table **??**) is exactly estimated as 0. As expected, when factor B is coded as a treatment contrast, the main effect of factor A estimates the effect of A nested within the baseline level of factor B.

Next, consider the main effect of factor B. According to the same logic, this main effect estimates the effect of factor B at the "location" where factor A is 0. Factor A is also coded as a treatment contrast, that is, it codes its baseline condition A1 as 0. The main effect of factor B estimates the effect of B nested within the baseline condition of A. Figure 7.1 shows that this effect should be 10; this indeed corresponds to the main effect of B as estimated in the regression model for treatment contrasts (see Table **??**, the *Estimate* for BB2). As we had seen before, the interaction term, however, does not differ between the treatment contrast and ANOVA ($t^2 = 2.24^2 = F = 5.00$).

How do we know what the "location" is, where a contrast applies? For the treatment contrasts discussed here, it is possible to reason this through because all contrasts are coded as 0 or 1. However,

how is it possible to derive the "location" in general? What we can do is to look at the hypotheses tested by the treatment contrasts (or the comparisons that are estimated) in the presence of an interaction between them by using the generalized matrix inverse. We go back to the default treatment contrasts. Then we extract the contrast matrix from the design matrix:

```
contrasts(df_contrasts4$A) <- contr.treatment(2)
contrasts(df_contrasts4$B) <- contr.treatment(2)
XcTr <- df_contrasts4 %>%
  group_by(A, B) %>%
  summarise() %>%
  model.matrix(~ 1 + A*B, .) %>%
  as.data.frame() %>% as.matrix()
rownames(XcTr) <- c("A1_B1","A1_B2","A2_B1","A2_B2")
XcTr
```

```
##       (Intercept) A2 B2 A2:B2
## A1_B1           1  0  0     0
## A1_B2           1  0  1     0
## A2_B1           1  1  0     0
## A2_B2           1  1  1     1
```

This shows the treatment contrast for factors `A` and `B`, and their interaction. We can now assign this contrast matrix to a `hypr` object. `hypr` automatically converts the contrast matrix into a hypothesis matrix, such that we can read from the hypothesis matrix which comparison are being estimated by the different contrasts.

```
htr <- hypr() # initialize empty hypr object
cmat(htr) <- XcTr # assign contrast matrix to hypr object
htr # look at the resulting hypothesis matrix
```

```
## hypr object containing 4 null hypotheses:
## H0.(Intercept): 0 = A1_B1
##           H0.A2: 0 = -A1_B1 + A2_B1
##           H0.B2: 0 = -A1_B1 + A1_B2
```

```
##       H0.A2:B2: 0 = A1_B1 - A1_B2 - A2_B1 + A2_B2
##
## Hypothesis matrix (transposed):
##       (Intercept) A2 B2 A2:B2
## A1_B1  1          -1 -1  1
## A1_B2  0           0  1 -1
## A2_B1  0           1  0 -1
## A2_B2  0           0  0  1
##
## Contrast matrix:
##       (Intercept) A2 B2 A2:B2
## A1_B1 1           0  0  0
## A1_B2 1           0  1  0
## A2_B1 1           1  0  0
## A2_B2 1           1  1  1
```

Note that the same result is obtained by applying the generalized inverse to the contrast matrix (this is what hypr does as well). An important fact is that when we apply the generalized inverse to the contrast matrix, we obtain the corresponding hypothesis matrix (for details see Schad et al., 2020b).

```
t(ginv2(XcTr))
```

```
##       (Intercept) A2 B2 A2:B2
## A1_B1  1          -1 -1  1
## A1_B2  0           0  1 -1
## A2_B1  0           1  0 -1
## A2_B2  0           0  0  1
```

As discussed above, the main effect of factor `A` estimates its effect nested within the baseline level of factor `B`. Likewise, the main effect of factor `B` estimates its effect nested within the baseline level of factor `A`.

How does this work for sum contrasts? They do not have a baseline condition that is coded as 0. In sum contrasts, however, the average of the contrast coefficients is 0. Therefore, main effects estimate

the average effect across factor levels. This is what is typically also tested in standard ANOVA. Let's look at the example shown in Table 7.2: given that factor B has a sum contrast, the main effect of factor A is tested as the average across levels of factor B. Figure 7.1 shows that the effect of factor A in level B1 is $10 - 10 = 0$, and in level B2 it is $20 - 40 = -20$. The average effect across both levels is $(0 - 20)/2 = -10$. Due to the sum contrast coding, we have to divide this by 2, yielding an expected effect of $-10/2 = -5$. This is exactly what the main effect of factor A measures (see Table 7.2, *Estimate* for A1).

Similarly, factor B tests its effect at the location 0 of factor A. Again, 0 is exactly the mean of the contrast coefficients from factor A, which is coded as a sum contrast. Therefore, factor B tests the effect of B averaged across factor levels of A. For factor level A1, factor B has an effect of $10 - 20 = -10$. For factor level A2, factor B has an effect of $10 - 40 = -30$. The average effect is $(-10 - 30)/2 = -20$, which again needs to be divided by 2 due to the sum contrast. This yields exactly the estimate of -10 that is also reported in Table 7.2 (*Estimate* for B1).

Again, we look at the hypothesis matrix for the main effects and the interaction:

```
contrasts(df_contrasts4$A) <- contr.sum(2)
contrasts(df_contrasts4$B) <- contr.sum(2)
XcSum <- df_contrasts4 %>%
  group_by(A, B) %>%
  summarise() %>%
  model.matrix(~ 1 + A*B, .) %>%
  as.data.frame() %>% as.matrix()
rownames(XcSum) <- c("A1_B1","A1_B2","A2_B1","A2_B2")

hsum <- hypr() # initialize empty hypr object
cmat(hsum) <- XcSum # assign contrast matrix to hypr object
hsum # look at the resulting hypothesis matrix
```

```
## hypr object containing 4 null hypotheses:
```

```
## H0.(Intercept): 0 = 1/4*A1_B1 + 1/4*A1_B2 + 1/4*A2_B1 + 1/4*A2_B2
##           H0.A1: 0 = 1/4*A1_B1 + 1/4*A1_B2 - 1/4*A2_B1 - 1/4*A2_B2
##           H0.B1: 0 = 1/4*A1_B1 - 1/4*A1_B2 + 1/4*A2_B1 - 1/4*A2_B2
##        H0.A1:B1: 0 = 1/4*A1_B1 - 1/4*A1_B2 - 1/4*A2_B1 + 1/4*A2_B2
##
## Hypothesis matrix (transposed):
##       (Intercept) A1   B1   A1:B1
## A1_B1  1/4         1/4  1/4  1/4
## A1_B2  1/4         1/4 -1/4 -1/4
## A2_B1  1/4        -1/4  1/4 -1/4
## A2_B2  1/4        -1/4 -1/4  1/4
##
## Contrast matrix:
##       (Intercept) A1 B1 A1:B1
## A1_B1  1           1  1  1
## A1_B2  1           1 -1 -1
## A2_B1  1          -1  1 -1
## A2_B2  1          -1 -1  1
```

This shows that each of the main effects now does not compute nested comparisons any more, but that they rather test their effect averaged across conditions of the other factor. The averaging involves using weights of 1/2. Moreover, the regression coefficients in the sum contrast measure half the distance between conditions, leading to weights of $1/2 \cdot 1/2 = 1/4$.

The general rule to remember from these examples is that when interactions between contrasts are estimated, what a main effect of a factor estimates depends on the contrast coding of the other factors in the design! The main effect of a factor estimates the effect nested within the location zero of the other contrast(s) in an analysis. If another contrast is centered, and zero is the average of this other contrast's coefficients, then the contrast of interest tests the average effect, averaged across the levels of the other factor. Importantly, this property holds only when the interaction between two contrasts is included into a model. If the interaction is omitted and only main effects are estimated, then there is no such "action at a distance".

This may be a very surprising result for interactions of contrasts. However, it is also essential to interpreting contrast coefficients involved in interactions. It is particularly relevant for the analysis of the default treatment contrast, where the main effects estimate nested effects rather than average effects.

7.2 One factor and one covariate

7.2.1 Estimating a group-difference and controlling for a covariate

In this section we treat the case where there are again two predictor variables for one dependent variable, but where one predictor variable is a discrete factor, and the other is a continuous covariate. Let's assume we have measured some response time (RT), e.g. in a lexical decision task. We want to predict the response time based on each subject's IQ, and we expect that higher IQ leads to shorter response times. Moreover, we have two groups of each 30 subjects. These are coded as factor F, with factor levels F1 and F2. We assume that these two groups have obtained different training programs to optimize their response times on the task. Group F1 obtained a control training, whereas group F2 obtained a training to improve lexical decisions. We want to test whether the training for better lexical decisions in group F2 actually leads to shorter response times compared to the control group F1. This is our main question of interest here, i.e., whether the training program in F2 leads to faster response times compared to the control group F1. We load the data, which is an artificially simulated data set.

```
data("df_contrasts5")
str(df_contrasts5)
```

```
## tibble[,4] [60 x 4] (S3: tbl_df/tbl/data.frame)
##  $ F : Factor w/ 2 levels "F1","F2": 1 1 1 1 1 1 1 1 1 1 ...
##  $ RT: num [1:60] 247 226 173 229 226 ...
```

```
##  $ IQ: num [1:60] 80.6 93.5 72 72.4 73.4 ...
##  $ id: Factor w/ 60 levels "1","2","3","4",..: 1 2 3 4 5 6 7 8 9 10 ...
```

Our main effect of interest is the factor F. We want to test its effect on response times and code it using scaled sum contrasts, such that negative parameter estimates would yield support for our hypothesis that response times are faster in the training group F2:

```
(contrasts(df_contrasts5$F) <- c(-0.5, +0.5))
```

```
## [1] -0.5  0.5
```

We run a linear model to estimate the effect of factor F, i.e., how strongly the response times in the two groups differ from each other.

```
fit_RT_F <- lm(RT ~ 1 + F,
               data = df_contrasts5)
```

```
round(summary(fit_RT_F)$coefficients)
```

```
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)      212          5      41        0
## F1               -25         10      -2        0
```

We find (see model estimates and data shown in Fig. 7.2) that response times in group F2 are roughly 25 ms faster than in group F1 (Estimate of −24). The 95% confidence intervals do not overlap with zero. This suggests that as expected, the training program that group F2 obtained seems to be successful in speeding up response times. We could now run a Bayes factor analysis on this data set to directly test this hypothesis, and maybe this would provide evidence for a difference in response times between groups.

However, let's assume we have allocated subjects to the two groups randomly. Let's say that we also measured the IQ of each person

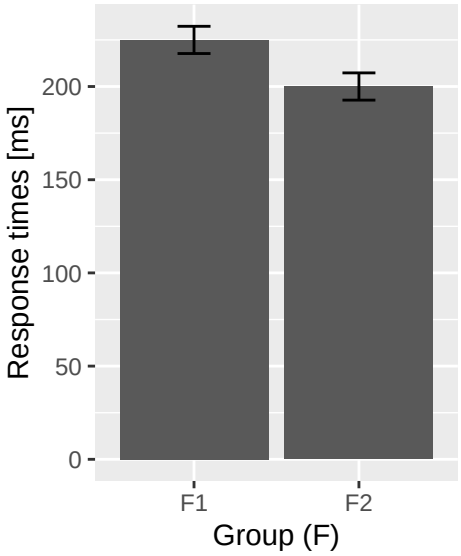


FIGURE 7.2: Means and error bars (showing standard errors) for a simulated data-set of response times for two different groups of subjects, who have obtained a training in lexical decisions (F2) versus have obtained a control training (F1).

using an IQ test. We did so, because we expected that IQ could have a strong influence on response times, and we wanted to control for this influence. We now can check whether the two groups had the same average IQ.

```
df_contrasts5 %>% group_by(F) %>% summarize(M.IQ = mean(IQ))
```

```
## # A tibble: 2 x 2
##   F      M.IQ
##   <fct> <dbl>
## 1 F1     85.
## 2 F2    115
```

Interestingly, group F2 did not only obtain an additional training and had faster response times, but group F2 also had a higher IQ (mean of 115) on average than group F1 (mean IQ = 85). Thus, the random allocation of subjects to the two groups seems

to have created - by chance - a difference in IQs. Now we can ask the question: why may response times in group F2 be faster than in group F1? Is this because of the training program in F2? Or is this simply because the average IQ in group F2 was higher than in group F1? To investigate this question, we add both predictor variables simultaneously in a linear model. Before we enter the continuous IQ variable, we center it, by subtracting its mean. Centering covariates is generally good practice. Moreover, it is often important to z-transform the covariate, i.e., to not only subtract the mean, but also to divide by its standard deviation (this can be done as follows: `df_contrasts5$IQ.s <- scale(df_contrasts5$IQ)`). The reason why this is often important is that the estimation doesn't work well if predictors have different scales. For the simple models we use here, the estimation works fine without z-transformation. However, for more realistic more complex models, z-transformation of covariates is often very important.

```
df_contrasts5$IQ.c <- df_contrasts5$IQ - mean(df_contrasts5$IQ)
fit_RT_F_IQ <- lm(RT ~ 1 + F + IQ.c,
                  data = df_contrasts5)
```

```
round(summary(fit_RT_F_IQ)$coefficients,2)
```

```
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)   212.50       4.77   44.51     0.00
## F1              7.00      13.62    0.51     0.61
## IQ.c           -1.07       0.32   -3.30     0.00
```

The results from the brms model now show that the difference in response times between groups (i.e., factor F) is not estimated to be −25 ms any more, but instead, the estimate is about +7 ms, and the 95% confidence intervals strongly overlap with zero (−20 to 33). Thus, it looks as if the groups would not differ from each other any more. At the same time, we see that the predictor variable IQ shows a negative effect (Estimate = −1 with 95% confidence

interval: −1.7 to −0.4), suggesting that - as expected - response times seem to be faster in subjects with higher IQ.

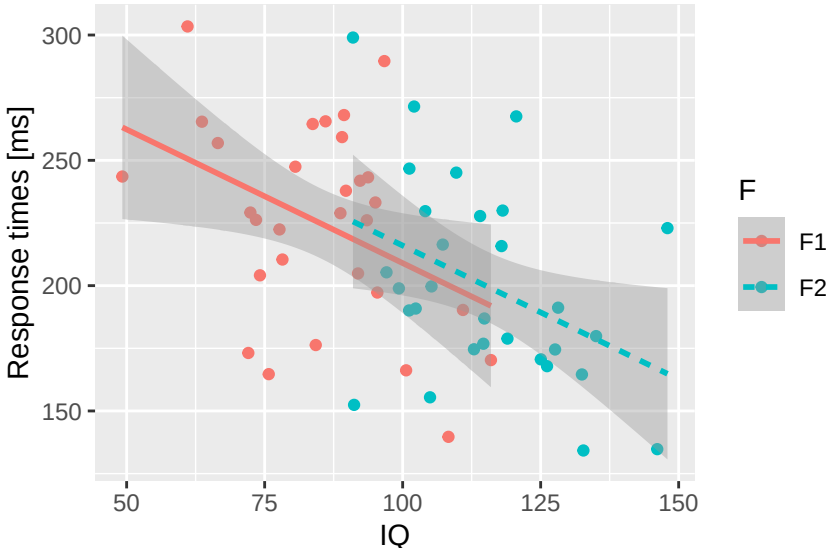


FIGURE 7.3: Response times as a function of individual IQ for two groups with a lexical decision training (F2) versus a control training (F1). Points indicate individual subjects, and lines with error bands indicate linear regression lines.

This result can also be seen in Figure 7.3, which shows that response times decrease with increasing IQ, as suggested by the linear model. However, the heights of the two regression lines do not differ from each other, consistent with the observation in the brms model that the effect of factor F did not seem to differ from zero. That is, factor F in the linear model estimates the difference in height of the regression line between both groups. That the height does not differ and the effect of F is estimated close to zero suggests that in fact group F2 showed faster response times not because of their additional training program. Instead, they had faster response times simply because their IQ was by chance higher on average compared to the control group F1. This analysis is called "analysis of covariance" (ANCOVA), where it's possible to test a group-difference after "controlling for" the influence of a covariate.

Importantly, we can see in Figure 7.3 that the two regression lines for the two groups are exactly parallel to each other. That is, the influence of IQ on response times seems to be exactly the same in both groups. This is actually a prerequiste for the ANCOVA analysis that needs to be checked in the data. That is, if we want to test the difference between groups after controlling for a covariate (here IQ), we have to test whether the influence of the covariate is the same in both groups. We can investigate this by including an interaction term between the factor and the covariate in the brms model:

```
fit_RT_FxIQ <- lm(RT ~ 1 + F * IQ.c,
                  data = df_contrasts5)
```

```
round(summary(fit_RT_FxIQ)$coefficients,2)
```

```
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)   212.50       6.87   30.93     0.00
## F1              7.00      13.74    0.51     0.61
## IQ.c           -1.07       0.33   -3.27     0.00
## F1:IQ.c         0.00       0.65    0.00     1.00
```

The estimate for the interaction (the term "F1:IQ.c") is very small here (close to 0) and the 95% confidence intervals clearly overlap with zero, showing that the two regression lines are estimated to be very similar, or parallel, to each other. If this is the case, then it is possible to correct for IQ when testing the group difference.

7.2.2 Estimating differences in slopes

We now take a look at a different data set.

```
data("df_contrasts6")
levels(df_contrasts6$F) <- c("simple","complex")
str(df_contrasts6)
```

```
## tibble[,4] [60 x 4] (S3: tbl_df/tbl/data.frame)
##  $ F : Factor w/ 2 levels "simple","complex": 1 1 1 1 1 1 1 1 1 1 ...
##  $ RT: num [1:60] 223 200 152 206 203 ...
##  $ IQ: num [1:60] 99.3 109.5 76.3 87.1 87.6 ...
##  $ id: Factor w/ 60 levels "1","2","3","4",..: 1 2 3 4 5 6 7 8 9 10 ...
```

This again contains data from response times (RT) in two groups. Let's assume the two groups have performed two different response time tasks, where one simple RT task doesn't rely on much cognitive processing (group "simple"), whereas the other task is more complex and depends on complex cognitive operations (group "complex"). We therefore expect that RTs in the simple task should be independent of IQ, whereas in the complex task, individuals with a high IQ should be faster in responding compared to individuals with low IQ. Thus, our primary hypothesis of interest states that the influence of IQ on RT differs between conditions. This means that we are interested in the difference between slopes. A slope in a linear regression assesses how strongly the dependent variable (here RT) changes with an increase of one unit on the covariate (here IQ), it thus assesses how "steep" the regression line is. Our research hypothesis is that the regression lines differ between groups.

The results, displayed in Figure 7.4, suggest that the data are consistent with our research hypothesis. For the subjects performing the complex task, response times seem to decrease with increasing IQ, whereas for subjects performing the simple task, response times seem to be independent of IQ. As stated before, our primary hypothesis relates to the difference in slopes. Statistically speaking, this is assessed in the interaction between the factor and the covariate. Thus, we run a linear model where the interaction is included. Importantly, we first use scaled sum contrasts for the group effect, and again center the covariate IQ.

```
contrasts(df_contrasts6$F) <- c(-0.5, +0.5)
df_contrasts6$IQ.c <- df_contrasts6$IQ - mean(df_contrasts6$IQ)
```

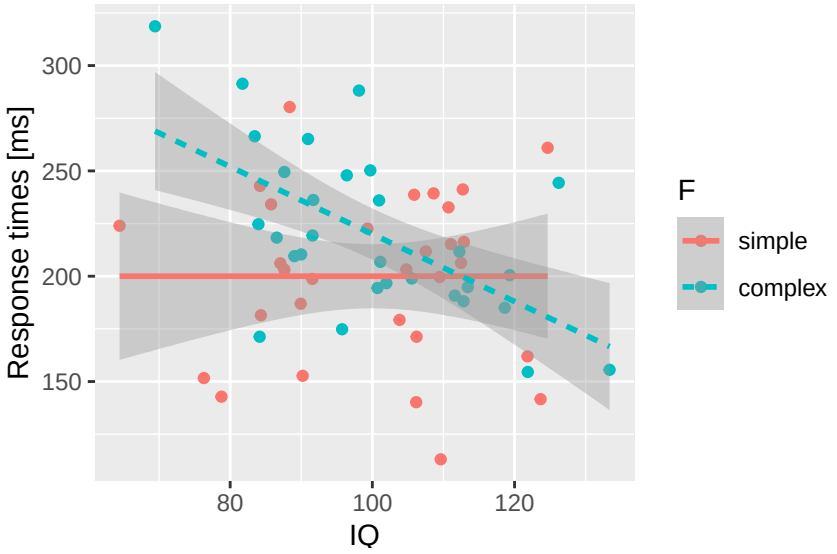


FIGURE 7.4: Response times as a function of individual IQ for two groups performing a simple versus a complex task. Points indicate individual subjects, and lines with error bands indicate linear regression lines.

```
fit_RT_FxIQ2 <- lm(RT ~ 1 + F * IQ.c,
                   data = df_contrasts6)
```

```
round(summary(fit_RT_FxIQ2)$coefficients,2)
```

```
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)    210.0       4.76   44.13     0.00
## F1              20.0       9.52    2.10     0.04
## IQ.c            -0.8       0.32   -2.48     0.02
## F1:IQ.c         -1.6       0.65   -2.48     0.02
```

We can see that the main effect of IQ (term "IQ.c") is negative (-0.8) with 95% confidence intervals -1.5 to -0.2, suggesting that overall response times decrease with increasing IQ. However, this is qualified by the interaction term, which is estimated to be negative

(−1.6), with 95% confidence intervals −2.9 to −0.3. This suggests that the slope in the complex group (which was coded as +0.5 in the scaled sum contrast) is more negative than the slope in the simple group (which was coded as −0.5 in the scaled sum contrast). Thus, the interaction assesses the difference between slopes.

We can also run a model where the simple slopes are estimated, i.e., the slope of IQ in the simple group and the slope of IQ in the complex group. This can be implemented by using the nested coding that we learned about in the previous section:

```
fit_RT_FnIQ2 <- lm(RT ~ 1 + F / IQ.c,
                   data = df_contrasts6)
```

```
round(summary(fit_RT_FnIQ2)$coefficients,2)
```

```
##                Estimate Std. Error t value Pr(>|t|)
## (Intercept)       210.0       4.76   44.13     0.00
## F1                 20.0       9.52    2.10     0.04
## Fsimple:IQ.c        0.0       0.46    0.00     1.00
## Fcomplex:IQ.c      -1.6       0.46   -3.51     0.00
```

Now we see that the slope of IQ in the simple group ("Fsimple:IQ.c") is estimated to be 0, with confidence intervals clearly including zero. By contrast, the slope in the complex group ("Fcomplex:IQ.c") is estimated as −1.6 (95% confidence interval −2.5 to −0.7). This is consistent with our hypothesis that high IQ speeds up response times for the complex but not for the simple task. We can also see from the nested analysis that the difference in slopes between conditions is $-1.6 - 0.0 = -1.6$. This is exactly the value for the interaction term that we estimated in the previous model, demonstrating that interaction terms assess the difference between slopes; i.e., they estimate the extent to which the regression lines in the two conditions are parallel, with an estimate of 0 indicating perfectly parallel lines.

A note: It is very important to always center covariates before

including them into a model. If covariates are not centered, then the main effects (here the main effect for the factor) cannot be interpreted as main effects any more.

Interestingly, one can also do analyses with interactions between a covariate and a factor, but by using different contrast codings. For example, if we use treatment contrasts for the factor, then the main effect of IQ.c assess not the average slope of IQ.c across conditions, but instead the nested slope of IQ.c within the baseline group of the treatment contrast. The interaction still assesses the difference in slopes between groups. In a situation where there are more than two groups, when one estimates the interaction of contrasts with a covariate, then the contrasts define which slopes are compared with each other in the interaction terms. For example, when using sum contrasts in an example where the influence of IQ is measured on response times for nouns, verbs, and adjectives, then there are two interaction terms: these assess (1) whether the slope of IQ for nouns is different from the average slope across conditions, and (2) whether the slope of IQ for verbs is different from the average slope across conditions. If one uses repeated contrasts in a situation where the influence of IQ on response times is estimated for word frequency conditions "low", "medium-low", "medium-high", and "high", then there are three interaction terms (one for each contrast). The first interaction term estimates the difference in slopes between "low" and "medium-low" word frequencies, the second interaction term estimates the difference in slopes between "medium-low" and "medium-high" word frequencies, and the third interaction term estimates the difference in slopes between "medium-high" and "high" word frequency conditions. Thus, the logic of how contrasts specify certain comparisons between conditions extends directly to the situation where differences in slopes are estimated.

7.3 Interactions in generalized linear models (with non-linear link functions)

We next look at generalized linear models, where a linear predictor is passed through a non-linear link function to predict the dependent variable. Examples for generalized linear models include logistic regression models and models assuming a log-normal or a Poisson distribution. Here, we treat an example with a logistic model in a 2 x 2 factorial between-subject design. The logistic model has the following non-linear link function: $p(y = 1 \mid x, b) = \frac{1}{1+\exp(-\eta)}$, where η is the latent linear predictor. For example, in our 2 x 2 factorial design with main effects A and B and their interaction, η is computed as a linear combination of the intercept plus the main effects and their interaction: $\eta = 1 + \beta_A x_A + \beta_B x_B + \beta_{A\times B} x_{A\times B}$.

Thus, there is a latent level of linear predictions (η), which are then passed through a non-linear link function to predict the probability that the observed data is a success ($p(y = 1)$). We will use this logistic model to analyze an example data set where the dependent variable is dichotomous, coded as either a 1 (indicating success) or a 0 (indicating failure).

We load a simulated data set where the dependent variable codes whether a subject performed a task successfully (pDV = 1) or not (pDV = 0). Moreover, the data set has two between-subject factors A and B. The means and frequentist 95% confidence intervals for each of the four conditions are shown in Table 7.3.

```
data("df_contrasts7")
str(df_contrasts7)
```

```
## tibble[,4] [200 x 4] (S3: tbl_df/tbl/data.frame)
##  $ A  : Factor w/ 2 levels "A1","A2": 1 1 1 1 1 1 1 1 1 1 ...
##  $ B  : Factor w/ 2 levels "B1","B2": 1 1 1 1 1 1 1 1 1 1 ...
##  $ pDV: int [1:200] 0 0 0 1 0 0 0 0 0 0 ...
##  $ id : int [1:200] 1 2 3 4 5 6 7 8 9 10 ...
```

TABLE 7.3: Summary statistics per condition for the simulated data.

Factor A	Factor B	N data	Means
A1	B1	50	0.2
A1	B2	50	0.5
A2	B1	50	0.2
A2	B2	50	0.8

To analyze this data, we use scaled sum contrasts, as we had done above for the 2×2 design with response times as the dependent variable, and which allow us to interpret the main effects as main effects. Next, we fit a linear model. The model specification is the same as the model with response times - with two differences: First, the `family` argument is now specified as `family = binomial(link = "logit")` to indicate the logistic model.

```
contrasts(df_contrasts7$A) <- c(-0.5,+0.5)
contrasts(df_contrasts7$B) <- c(-0.5,+0.5)
# GLM
fit_pDV_AB.sum <-glm(pDV ~ 1 + A*B,
                  data = df_contrasts7,
                  family = binomial(link = "logit"))
```

```
round(summary(fit_pDV_AB.sum)$coefficients,2)
```

```
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept)    -0.35       0.17   -2.06     0.04
## A1              0.69       0.34    2.06     0.04
## B1              2.08       0.34    6.17     0.00
## A1:B1           1.39       0.67    2.06     0.04
```

The results from this analysis show that the estimates for the two main effects ("A1" and "B1") as well as the interaction ("A1:B1") are positive and the 95% confidence intervals do not include zero.

We could now proceed to perform likelihood ratio tests to investigate the evidence that there is for each of the effects.

7.4 Summary

To summarize, we have seen interesting results for contrasts in the context of 2×2 designs, where depending on the contrast coding, the factors estimated nested effects (treatment contrasts) or main effects (sum contrasts). We also saw that it is possible to code contrasts for a 2×2 design, by creating one large factor comprising all design cells, and by specifying all effects of interest in one large contrast matrix. In designs with one factor and one covariate it is possible to control group differences for differences in the covariate (ANCOVA), or to test whether regression slopes are parallel in different experimental conditions. Finally, contrast coding in generalized linear models works the same way as in the standard linear model.

8

Using simulation to understand your model

Data analysis is often taught as if the goal is to work out the p-value and make a decision: reject or fail to reject the null hypothesis. However, understanding the long-run properties of one's experiment design and statistical model under repeated sampling requires more work and thought. Specifically, it is important to understand (a) what one's model's power and Type I error properties are, and (b) whether the model we plan to fit to our data can, even in principle, recover the parameters in the model.

In order to study these properties of one's model, it is necessary to learn to simulate data that reflects our experimental design. Let's think about how to simulate data given a Latin-square 2 condition repeated measures design. We begin with our familiar running example, the Grodner and Gibson (2005) English relative clause data.

8.1 A reminder: The maximal linear mixed model

Recall the structure of the linear mixed model that can be used to fit the Grodner and Gibson (2005) data. We will discuss the so-called maximal model here—varying intercepts and slopes for subject and for item, with correlations—because that is the most general case.

In the model specification below, i indexes subjects, j items. The vector `so` has the sum contrast coding as usual: object relatives are coded as +1/2 and subject relatives as -1/2. We use this coding instead of ± 1 as before, because now the slope will reflect the

effect size rather than two times the effect size (see the hypothesis testing chapter).

Every row in the data-frame can be uniquely identified by the subject and item id, because this is a Latin square design and each subject sees exactly one instance of each item in a particular condition.

$$y_{ij} = \beta_0 + u_{0i} + w_{0j} + (\beta_1 + u_{1i} + w_{1j}) \times so_{ij} + \varepsilon_{ij} \tag{8.1}$$

where $\varepsilon_{ij} \sim Normal(0, \sigma)$ and

$$\Sigma_u = \begin{pmatrix} \sigma^2_{u0} & \rho_u \sigma_{u0} \sigma_{u1} \\ \rho_u \sigma_{u0} \sigma_{u1} & \sigma^2_{u1} \end{pmatrix} \quad \Sigma_w = \begin{pmatrix} \sigma^2_{w0} & \rho_w \sigma_{w0} \sigma_{w1} \\ \rho_w \sigma_{w0} \sigma_{w1} & \sigma^2_{w1} \end{pmatrix} \tag{8.2}$$

$$\begin{pmatrix} u_0 \\ u_1 \end{pmatrix} \sim \mathcal{N}\left(\begin{pmatrix} 0 \\ 0 \end{pmatrix}, \Sigma_u \right), \quad \begin{pmatrix} w_0 \\ w_1 \end{pmatrix} \sim \mathcal{N}\left(\begin{pmatrix} 0 \\ 0 \end{pmatrix}, \Sigma_w \right) \tag{8.3}$$

β_0 and β_1 are the intercept and slope, representing the grand mean and the deviation from the grand mean in each condition. u are the subject level adjustments, and w the item level adjustments to the intercept and slope.
The above mathematical model expresses a generative process. In order to produce simulated data using the above process, we have to decide on some parameter values. We do this by estimating the parameters from the Grodner and Gibson (2005) study.

8.2 Obtain estimates from a previous study

First we load and prepare the relative clause data for data analysis.

```
gg05e1<-read.table("data/grodnergibsonE1crit.txt",header=TRUE)


gg05e1$so <- ifelse(gg05e1$condition=="objgap",1/2,-1/2)
gg05e1$logrt<-log(gg05e1$rawRT)
```

Next, fit the so-called maximal model. We will ignore the singularity warning as it won't affect us in our simulations.

```
library(lme4)
m<-lmer(logrt ~ so +
          (1+so|subject) +
          (1+so|item),
        data=gg05e1,
        ## "switch off" warnings:
        control=lmerControl(calc.derivs=FALSE))
```

The model summary shows that we can reject the null hypothesis of no difference in relative clauses:

```
summary(m)$coefficients
```

```
##             Estimate Std. Error    df t value
## (Intercept)    5.883    0.05202 43.07 113.082
## so             0.124    0.04932 31.28   2.515
##              Pr(>|t|)
## (Intercept) 6.384e-55
## so          1.726e-02
```

Let's focus on that effect for our power analysis. What is the prospective power of detecting this effect *for a future study*? Note that we never compute power for an existing study—that is called post-hoc power and is a pointless quantity to compute because once the p-value is known, the power is just a transformation of the p-value (Hoenig and Heisey, 2001).

What we are doing below will *look* like post-hoc power because we are using existing data to compute power. However, what is crucially different in our approach is that (a) we remain unsure about the true effect, (b) we are making a statement about what the power properties would be *if we ran the same study again*, with new subjects, but in the same environment (lab, etc.). We are not making any claim about the power properties of the *current* experiment; that ship has already sailed, the data are already at hand! Once the data are analyzed, it's too late to compute power for that particular data-set. A power analysis is only relevant for a design to be run in the future.

8.3 Decide on a range of plausible values of the effect size

Notice that the effect in milliseconds is relatively large, given the estimates from similar phenomena in reading studies in psycholinguistics (Jäger et al., 2017):

```
b0<-summary(m)$coefficients[1,1]
b1<-summary(m)$coefficients[2,1]
## effect estimate in log ms:
b1
```

```
## [1] 0.124
```

```
## effect estimate in ms:
exp(b0+b1*(0.5))-exp(b0+b1*(-0.5))
```

```
## [1] 44.54
```

But the standard errors tell us that the effect could be as small or as large as the following values:

```
b1_stderr<-summary(m)$coefficients[2,2]
lower<-b1-2*b1_stderr
upper<-b1+2*b1_stderr
lower;upper
```

```
## [1] 0.02539
```

```
## [1] 0.2227
```

The above range 0.03 and 0.22 arises because the range of plausible effect sizes is between $\hat{\beta}_1 \pm 2SE$ on the log ms scale.

On the ms scale, the range is:

```
exp(b0+lower*(0.5))-exp(b0+lower*(-0.5))
```

```
## [1] 9.113
```

```
exp(b0+upper*(0.5))-exp(b0+upper*(-0.5))
```

```
## [1] 80.09
```

On the ms scale we see that that's a *lot* of uncertainty in the effect size! With some experience, you will come to recognize that such a wide confidence bound is a sign of low power. We will just establish the prospective power properties of this study in a minute.

We can take the above uncertainty of the $\hat{\beta}_1$ estimator into account (on the log ms scale—remember that the model is based on log rt) by assuming that the effect has the following uncertainty on the log ms scale:

$$\beta_1 \sim Normal(0.12, 0.05) \tag{8.4}$$

Here, we are doing something that is, strictly speaking, Bayesian in thinking. We are describing our uncertainty about the true effect from the best estimate we have—existing data. To talk about the

uncertainty, we are (ab)using the 95% confidence interval (treating it like its telling us the range of plausible values). Recall that strictly speaking, in the frequentist paradigm, one cannot talk about the probability distribution of the effect size—in frequentist theory, the true value of the parameter is a point value, it has no distribution. The range $\hat{\beta}_1 \pm 2 \times SE$ refers to the estimated mean of the sampling distribution of the sample means, and to the standard deviation of this sampling distribution. Thus, strictly speaking, this range does not reflect our uncertainty about the true parameter's value. Having said this, we are going to use the effect estimates from our model fit as a starting point for our power analysis because this is the best information we have so far about the English relative clause design.

8.4 Extract parameter estimates

Next, in preparation for the power analysis, we extract all the parameter estimates from the model we have fit above. The parameters are:

- The two fixed effects (the β parameters)
- The residuals' standard deviation
- The standard deviations of the subject intercept and slope adjustments, and the corresponding correlation matrix.
- The standard deviations of the item intercept and slope adjustments, and the corresponding correlation matrix.

The correlation matrices and the subject/item random effects standard deviations are used to assemble the variance covariance matrix; this is done using the `sdcor2cov` function from the `SIN` package; recall the discussion in chapter 1. For the variance covariance matrix for items random effects, we use an intermediate value of 0.5 for the correlation parameter because the linear mixed model was unable to estimate the parameter.

```
m<-lmer(logrt ~ so +
          (1+so|subject) +
          (1+so|item),
        data=gg05e1,
        ## "switch off" warnings:
        control=lmerControl(calc.derivs=FALSE))

## extract parameter estimates:
beta<-round(summary(m)$coefficients[,1],4)
sigma_e<-round(attr(VarCorr(m),"sc"),2)
subj_ranefsd<-round(attr(VarCorr(m)$subject,"stddev"),4)
subj_ranefcorr<-round(attr(VarCorr(m)$subject,"corr"),1)

## assemble variance-covariance matrix for subjects:
Sigma_u<-SIN::sdcor2cov(stddev=subj_ranefsd,
                        corr=subj_ranefcorr)

item_ranefsd<-round(attr(VarCorr(m)$item,"stddev"),4)
item_ranefcorr<-round(attr(VarCorr(m)$item,"corr"),1)

## assemble variance matrix for items:

## choose some intermediate values for correlations:
corr_matrix<-(diag(2) + matrix(rep(1,4),ncol=2))/2

Sigma_w<-SIN::sdcor2cov(stddev=item_ranefsd,
                        corr=corr_matrix)
```

8.5 Define a function for generating data

Next, we define a function that generates repeated measures data given the parameter estimates. The basic idea here is the following.

- First, create a data-frame that represents a Latin-square design.

- Then, given the condition id, and the subject and item ids in each row of the data frame, generate data row-by-row.

We explain these steps next.

8.5.1 Generate a Latin-square design

First, consider how one can create a Latin-square design. Suppose we have four items and four subjects. For such an experiment, we would create two groups, g1 and g2, with the following layout.

```
nitem<-4
nsubj<-4

g1<-data.frame(item=1:nitem,
               cond=rep(c("a","b"),nitem/2))
g2<-data.frame(item=1:nitem,
               cond=rep(c("b","a"),nitem/2))
g1; g2
```

```
##   item cond
## 1    1    a
## 2    2    b
## 3    3    a
## 4    4    b
```

```
##   item cond
## 1    1    b
## 2    2    a
## 3    3    b
## 4    4    a
```

Half the total number of subjects will be assigned to group 1 and half to group 2:

```
## assemble data frame:
gp1<-g1[rep(seq_len(nrow(g1)),
            nsubj/2),]
```

```
  gp2<-g2[rep(seq_len(nrow(g2)),
              nsubj/2),]

  simdat<-rbind(gp1,gp2)

  ## add subject column:
  simdat$subj<-rep(1:nsubj,each=nitem)
```

Finally, the contrast coding for each row in the data-frame is set up:

```
  ## add contrast coding:
  simdat$so<-ifelse(simdat$cond=="a",-1/2,1/2)
```

8.5.2 Generate data row-by-row

Then, we proceed row-by-row in this data frame, and generate data for each subject, item, and condition. For example, the first row of our simulated data-set has subject id:

```
simdat[1,]$subj
```

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

Similarly, the first row has item id:

```
simdat[1,]$item
```

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

The first row's condition coding is:

```
simdat[1,]$so
```

```
## [1] -0.5
```

These three pieces of information are what we need to generate data for the first row. Recall that the model for subject i and item j in condition `so` is

$$\beta_0 + u_{0i} + w_{0j} + (\beta + u_{1i} + w_{1j}) \times so + \varepsilon \text{ where } \varepsilon \sim N(0, \sigma) \quad (8.5)$$

The terms u0, w0, and u1, w1, which are the adjustments to the intercepts and slopes by subject and item, are stored in two matrices that are generated randomly each time that we simulate new subjects/items. Recall from chapter 1 how bivariate data are generated. The intercept and slope adjustments will be generated using the `mvrnorm` function in the MASS library. For example, given the variance covariance matrix for subjects `Sigma_u` that we created above, the subject random effects (intercept and slope adjustments) for 10 subjects can be generated in a matrix as follows:

```
library(MASS)
u<-mvrnorm(n=10,
           mu=c(0,0),Sigma=Sigma_u)
u
```

```
##       (Intercept)       so
##  [1,]     0.11016  0.28122
##  [2,]     0.83391  0.21912
##  [3,]    -0.12487  0.06087
##  [4,]    -0.02720 -0.06286
##  [5,]     0.42886  0.24292
##  [6,]    -0.09001 -0.24314
##  [7,]     0.06948  0.12048
##  [8,]     0.21241  0.04330
##  [9,]    -0.48465 -0.39614
## [10,]    -0.04833  0.35370
```

Each row in this matrix is the intercept and slope adjustment for a subject; the row number indexes the subject id. For example, subject 1's intercept adjustment is:

```
u[1,1]
```

```
## (Intercept)
##      0.1102
```

Subject 1's slope adjustment is:

```
u[1,2]
```

```
##     so
## 0.2812
```

Analogously to the subject random effects matrix, a matrix for items random effects is also generated. As an example, we generate simulated random effects for 10 items:

```
w<-mvrnorm(n=10,
             mu=c(0,0),Sigma=Sigma_w)
w
```

```
##             [,1]       [,2]
##  [1,]  0.020043  0.0733144
##  [2,]  0.040632  0.0099160
##  [3,]  0.002638 -0.1848623
##  [4,]  0.006454 -0.0252776
##  [5,]  0.024081  0.0068770
##  [6,] -0.068960 -0.1846818
##  [7,] -0.009174  0.0097644
##  [8,]  0.020528 -0.0007004
##  [9,] -0.047864  0.1347658
## [10,] -0.014866  0.0425107
```

Now, to generate simulated data for subject 1 and item 1 for object relatives, we simply need to run this line of code:

```
rlnorm(1,beta[1] +    u[1,1] + w[1,1] +
         (beta[2]+u[1,2]+w[1,2])*(0.5),
                   sigma_e)
```

```
## [1] 423.2
```

For subject 2, all that would change is the subject id: instead of `u[1,1]`, and `u[1,2]` we would write `u[2,1]` and `u[2,2]`. The for-loop below works through the `simdat` data-frame row by row, looks up the subject id, the item id for that row, and the condition coding for that row, and fills in the simulated reading time using the above code. This is how the simulated data are generated.

Here is the complete function for generating simulated data. It uses all the bits of code we discussed above.

```
library(MASS)
## assumes that no. of subjects and
## no. of items is divisible by 2.
gen_sim_lnorm2<-function(nitem=16,
                         nsubj=42,
                         beta=NULL,
                         Sigma_u=NULL, # subject vcov matrix
                         Sigma_w=NULL, # item vcov matrix
                         sigma_e=NULL){
  ## prepare data frame for a two-condition latin square:
  g1<-data.frame(item=1:nitem,
                 cond=rep(c("a","b"),nitem/2))
  g2<-data.frame(item=1:nitem,
                 cond=rep(c("b","a"),nitem/2))


  ## assemble data frame:
  gp1<-g1[rep(seq_len(nrow(g1)),
              nsubj/2),]
  gp2<-g2[rep(seq_len(nrow(g2)),
```

```
            nsubj/2),]

  simdat<-rbind(gp1,gp2)

  ## add subject column:
  simdat$subj<-rep(1:nsubj,each=nitem)

  ## add contrast coding:
  simdat$so<-ifelse(simdat$cond=="a",-1/2,1/2)

  ## subject random effects:
  u<-mvrnorm(n=length(unique(simdat$subj)),
             mu=c(0,0),Sigma=Sigma_u)

  ## item random effects
  w<-mvrnorm(n=length(unique(simdat$item)),
             mu=c(0,0),Sigma=Sigma_w)

  ## generate data row by row:
  N<-dim(simdat)[1]
  rt<-rep(NA,N)
  for(i in 1:N){
    rt[i] <- rlnorm(1,beta[1] +
                      u[simdat[i,]$subj,1] +
                      w[simdat[i,]$item,1] +
                      (beta[2]+u[simdat[i,]$subj,2]+
                         w[simdat[i,]$item,2])*simdat$so[i],
                    sigma_e)
  }
  simdat$rt<-rt
  simdat$subj<-factor(simdat$subj)
  simdat$item<-factor(simdat$item)
  simdat}
```

Let's generate some simulated data and check what the data look like:

```
dat<-gen_sim_lnorm2(nitem=16,
                            nsubj=42,
                           beta=beta,
                           Sigma_u=Sigma_u,
                           Sigma_w=Sigma_w,
                          sigma_e=sigma_e)
```

The data have the expected structure:

```
## fully  crossed subjects and items:
head(t(xtabs(~subj+item,dat)))
```

```
##      subj
## item 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19
##    1 1 1 1 1 1 1 1 1  1  1  1  1  1  1  1  1  1  1
##    2 1 1 1 1 1 1 1 1  1  1  1  1  1  1  1  1  1  1
##    3 1 1 1 1 1 1 1 1  1  1  1  1  1  1  1  1  1  1
##    4 1 1 1 1 1 1 1 1  1  1  1  1  1  1  1  1  1  1
##    5 1 1 1 1 1 1 1 1  1  1  1  1  1  1  1  1  1  1
##    6 1 1 1 1 1 1 1 1  1  1  1  1  1  1  1  1  1  1
##      subj
## item 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35
##    1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1
##    2  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1
##    3  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1
##    4  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1
##    5  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1
##    6  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1
##      subj
## item 36 37 38 39 40 41 42
##    1  1  1  1  1  1  1  1
##    2  1  1  1  1  1  1  1
##    3  1  1  1  1  1  1  1
##    4  1  1  1  1  1  1  1
##    5  1  1  1  1  1  1  1
##    6  1  1  1  1  1  1  1
```

```
##  8 measurements per condition:
head(t(xtabs(~subj+cond,dat)))
```

```
##     subj
## cond 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19
##    a 8 8 8 8 8 8 8 8 8  8  8  8  8  8  8  8  8  8  8
##    b 8 8 8 8 8 8 8 8 8  8  8  8  8  8  8  8  8  8  8
##     subj
## cond 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35
##    a  8  8  8  8  8  8  8  8  8  8  8  8  8  8  8  8
##    b  8  8  8  8  8  8  8  8  8  8  8  8  8  8  8  8
##     subj
## cond 36 37 38 39 40 41 42
##    a  8  8  8  8  8  8  8
##    b  8  8  8  8  8  8  8
```

```
##  contrast coding check:
xtabs(~so+cond,dat)
```

```
##       cond
## so       a   b
##   -0.5 336   0
##   0.5    0 336
```

```
## condition b is slower than a:
round(with(dat,tapply(rt,cond,mean)))
```

```
##   a   b
## 371 459
```

Everything checks out.

8.6 Repeated generation of data to compute power

With the function for simulating data ready, we are now able to repeatedly generated simulated data. Next, we generate data 100 times, fit a linear mixed model each time, and extract the t-value from each linear mixed model fit. The vector t-values is then used to compute power: we simply look at the proportion of absolute t-values that exceed the value 2.

```
nsim<-100
sotval<-rep(NA,nsim)

for(i in 1:nsim){
#generate sim data:
dat<-gen_sim_lnorm2(nitem=16,
                          nsubj=42,
                        beta=beta,
                        Sigma_u=Sigma_u,
                        Sigma_w=Sigma_w,
                       sigma_e=sigma_e)

## fit model to sim data:
m<-lmer(log(rt)~so+(1+so|subj)+(1+so|item),dat,
        control=lmerControl(calc.derivs=FALSE))
## extract the t-value
sotval[i]<-summary(m)$coefficients[2,3]
}

pow<-mean(abs(sotval)>2)
##save(pow,file="data/powerbeta1mean.Rda")
```

The power estimate from the above code is 0.97.

The following computation will take a lot of time because we are generating and fitting data 3×100 times. Here, we will assume that the true effect has the following range of plausible values:

```
## lower bound:
.12-2*.05
```

```
## [1] 0.02
```

```
## mean
.12
```

```
## [1] 0.12
```

```
## upper bound
.12+2*.05
```

```
## [1] 0.22
```

We will run two for-loops now: the first for-loop selects one of the plausible values as the value for the slope, and then the second for-loop runs 100 simulations to compute power for that effect size.

This time, instead of ignoring convergence problems, we can record the proportion of times that we get a convergence failure or problem, and we can discard that result:

```
nsim<-100
## effect size possibilities:
b1_est<-c(0.02,0.12,0.22)
sotvals<-matrix(rep(NA,nsim*length(b1_est)),ncol=nsim)
failed<-matrix(rep(0,nsim*length(b1_est)),ncol=nsim)
for(j in 1:length(b1_est)){
for(i in 1:nsim){
  beta[2]<-b1_est[j]
  dat_sim<-gen_sim_lnorm2(nitem=16,
                          nsubj=42,
                        beta=beta,
                        Sigma_u=Sigma_u,
                        Sigma_w=Sigma_w,
```

```
                        sigma_e=sigma_e)

## no correlations estimated to
## minimize convergence problems:
## analysis done after log-transforming:
m<-lmer(log(rt) ~ so + (so||subj) +
          (so||item), data=dat_sim)
## ignore failed trials
if(any( grepl("failed to converge", m@optinfo$conv$lme4$messages)) ||
        any(grepl("boundary", m@optinfo$conv$lme4$messages))){
  failed[j,i]<-1
} else{
sotvals[j,i]<-summary(m)$coefficients[2,3]
}}}
```

You can examine the proportion of failed convergences:

```
## proportion of convergence failures:
rowMeans(failed)
```

Power can now be computed for each effect size:

```
pow<-rep(NA,length(b1_est))
for(k in 1:length(b1_est)){
  pow[k]<-mean(abs(sotvals[k,])>2,na.rm=TRUE)
}

##save(pow,file="data/powerbeta1meanlowerupper.Rda")
```

The power estimate for the lower bound of the β_1 estimate is 0.09, and for the upper bound is 1. The power estimate for the mean estimate of β_1 is 0.73.

Notice that there is a lot of uncertainty about the power estimate here!

Recall that power is a **function** of

- effect size
- standard deviation(s); in linear mixed models, these are all the variance components and the correlations
- sample size (numbers of subjects and items)

In papers, you will often see text like "power was x%". This statement reflects a misunderstanding; power is best plotted as a function of (a subset of) these variables.

In the discussion above, we display a range of power estimates; this range reflects our uncertainty about the power estimate as a function of the plausible effect sizes. Often (as here) this uncertainty will be very high! I.e., given what one knows so far, it may be difficult to pin down what the assumed effect size etc., should be, and that makes it hard to give a precise range for your power estimate.

8.7 What you can now do

Given the above code and workflow, you can now figure out how many subjects you might need to achieve 80% power, assuming a certain effect size (or a range of effect sizes as above) and assuming some specific values for the standard deviations and variance covariance matrices.

You can also study Type I error properties of your model as a function of whether the model is a maximal model or a varying intercepts only model.

Example: Compute power as a function of effect size and sample size. Note that the number of subjects has to be even, otherwise the simulation code will fail! One could put in a test for this in the code: if the number of subjects is divisible by 2, the modulo function should return 0:

```
10%%2
```

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

```
11%%2
```

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

```
## define a function for computing power
## (as a function of effect size and
## subject sample size:
compute_power<-function(b=NULL,nsubjects=28){
if(nsubjects%%2!=0){stop("No. of subjects must be divisible by 2.")}
nsim<-100
sotvals<-rep(NA,nsim)
failed<-rep(0,nsim)
for(i in 1:nsim){
  beta[2]<-b
  dat_sim<-gen_sim_lnorm2(nitem=24,
                          nsubj=nsubjects,
                         beta=beta,
                         Sigma_u=Sigma_u,
                         Sigma_w=Sigma_w,
                        sigma_e=sigma_e)

## no correlations estimated to avoid convergence problems:
## analysis done after log-transforming:
m<-lmer(log(rt) ~ so + (so||subj) +
          (so||item), data=dat_sim)
## ignore failed trials
if(any( grepl("failed to converge", m@optinfo$conv$lme4$messages) )){
  failed[i]<-1
} else{
sotvals[i]<-summary(m)$coefficients[2,3]
}}
```

```
## power:
paste("Power:",round(mean(abs(sotvals)>2,
                          na.rm=TRUE),2),sep=" ")
}
```

```
## usage: this will halt function
## with an error message
# compute_power(b=0.03,nsubjects=29)

compute_power(b=0.03,nsubjects=28)
```

[1] "Power: 1"

8.8 Using the package designr to simulate data and compute power

There are major advantages to writing one's own code for generating repeated measures data: one can control exactly what one considers to be the generative process, and one can obtain a deeper understanding of how the linear mixed model is set up. However, for certain standard experimental designs, one can use pre-defined functions for simulating data. The advantage of using such packages is that one can abstract away from the implementation details; but this can also be a disadvantage because one might end up simulating data without really understanding the assumptions behind the data-generative process.

An R package, designr, provides some built-in functions that can simplify the process of computing power for standard repeated measures designs. For example, consider again the two-condition design involving English relative clauses that we saw earlier (Grodner and Gibson, 2005).

8.8.1 Simulating data with two conditions

First, load the data, and prepare the contrast coding and log-transform the reading times.

```
gg05e1<-read.table("data/grodnergibsonE1crit.txt",header=TRUE)
gg05e1$so <- ifelse(gg05e1$condition=="objgap",1/2,-1/2)
gg05e1$logrt<-log(gg05e1$rawRT)
```

Next, we fit a linear mixed model to the data. As an example, we use raw reading times on the millisecond scale.

```
## frequentist analysis:
library(lme4)
m_lmer<-lmer(rawRT~so + (1+so|subject)+(1+so|item),gg05e1)
```

```
## boundary (singular) fit: see ?isSingular
```

```
summary(m_lmer)
```

```
## Linear mixed model fit by REML. t-tests use
##   Satterthwaite's method [lmerModLmerTest]
## Formula:
## rawRT ~ so + (1 + so | subject) + (1 + so | item)
##    Data: gg05e1
##
## REML criterion at convergence: 9645
##
## Scaled residuals:
##    Min     1Q Median     3Q    Max
## -2.392 -0.337 -0.127  0.131 12.504
##
## Random effects:
##  Groups   Name        Variance Std.Dev. Corr
##  subject  (Intercept) 24152    155.4
##           so          27603    166.1    1.00
```

```
##  item     (Intercept)  2246     47.4
##           so           7663     87.5    1.00
##  Residual             89503    299.2
## Number of obs: 672, groups:  subject, 42; item, 16
##
## Fixed effects:
##             Estimate Std. Error    df t value Pr(>|t|)
## (Intercept)    420.2       29.1  43.6    14.4   <2e-16
## so             102.3       40.9  32.1     2.5    0.018
##
## Correlation of Fixed Effects:
##    (Intr)
## so 0.734
## optimizer (nloptwrap) convergence code: 0 (OK)
## boundary (singular) fit: see ?isSingular
```

The correlations between the varying intercepts and slopes could not be estimated, so the model is over-parameterized. We can refit the model without correlations:

```
m_lmer<-lmer(rawRT~so + (1+so||subject)+(1+so||item),gg05e1)
summary(m_lmer)
```

```
## Linear mixed model fit by REML. t-tests use
##   Satterthwaite's method [lmerModLmerTest]
## Formula:
## rawRT ~ so + (1 + so || subject) + (1 + so || item)
##    Data: gg05e1
##
## REML criterion at convergence: 9684
##
## Scaled residuals:
##    Min     1Q Median     3Q    Max
## -1.966 -0.325 -0.118  0.052 12.924
##
## Random effects:
##  Groups    Name        Variance Std.Dev.
```

```
##  subject   (Intercept) 23574    153.5
##  subject.1 so          22289    149.3
##  item      (Intercept)  1367     37.0
##  item.1    so           3635     60.3
##  Residual               93001    305.0
## Number of obs: 672, groups:  subject, 42; item, 16
##
## Fixed effects:
##             Estimate Std. Error    df t value Pr(>|t|)
## (Intercept)    420.2       28.0  40.1   15.00   <2e-16
## so             102.3       36.2  24.0    2.82   0.0094
##
## Correlation of Fixed Effects:
##    (Intr)
## so 0.000
```

Next, we generate a data-frame through the `designr` package, with the same design as the data used above:

```
library(designr)

## example
## define data frame:
design<-fixed.factor("so", levels=c("subjgap", "objgap")) +
  random.factor("subject", instances=42)+
  random.factor("item", instances=16)
dat <- design.codes(design)
dat$so <- ifelse(dat$so=="objgap",1/2,-1/2)
head(dat)
```

```
## # A tibble: 6 x 3
##   subject   item      so
##   <fct>     <fct>  <dbl>
## 1 subject01 item01  -0.5
## 2 subject01 item01   0.5
## 3 subject01 item02  -0.5
## 4 subject01 item02   0.5
```

```
## 5 subject01 item03  -0.5
## 6 subject01 item03   0.5
```

To compute prospective power based on the estimates from the English relative clause data, the following for-loop is run. In each iteration, data is first generated using the design set up above; this is done using the `simLMM` function from the `designr` package. Notice here that a "maximal" model is specified, but the correlations are set to zero (CP=0.0); this has the effect that the data generated assume no correlation between the intercept and slope for subjects and items. Having generated the data, the `lmer` function is used to fit the model to the simulated data, and then the t-value of the slope is saved. This t-value can then be used to compute power in the usual way.

```
tvals<-rep(NA,100)

for(i in 1:100){
## generate data:
dat$ysim <- simLMM(formula = ~ 1 + so + (1 + so | subject) +
                     (1+so|item),
                   data = dat,
                   LMM = m_lmer,
                   CP =  0.0,
                   empirical=FALSE,verbose=FALSE,
                   family="gaussian")

## fit model to simulated data:
m<-lme4::lmer(ysim ~ so + (so || subject) + (so||item),
              data=dat,
              control=lmerControl(calc.derivs=FALSE))
## save t-value
tvals[i]<-summary(m)$coefficients[2,3]
}

mean(abs(tvals)>2)
```

```
## [1] 0.91
```

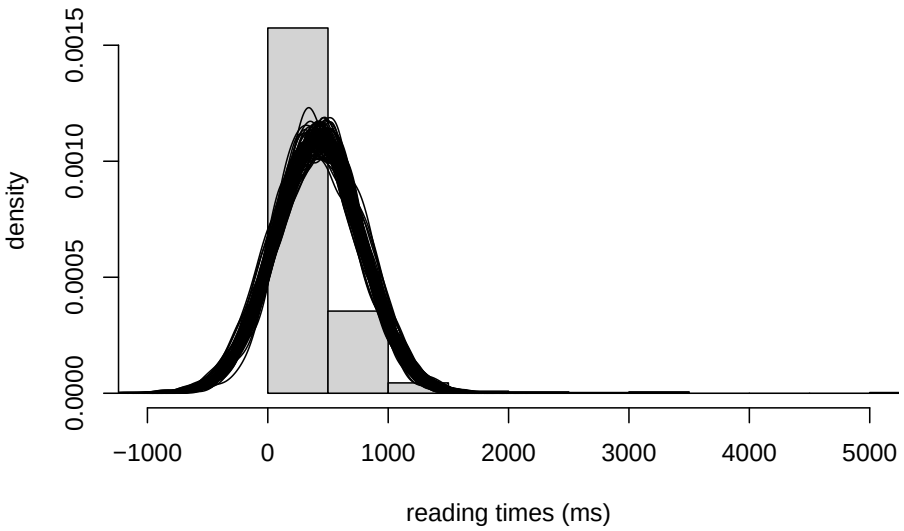
If we also want to examine the distribution of the data generated, and the distribution of the parameters under repeated sampling, we can easily record that information within the for-loop:

```
n<-dim(dat)[1]
nsim<-100
ysims<-matrix(rep(NA,n*nsim),ncol=nsim)

for(i in 1:100){
## generate data:
ysim <- simLMM(formula = ~ 1 + so + (1 + so | subject) + (1+so|item),
                   data = dat,
                   LMM = m_lmer,
                   CP =  0.0,
                   empirical=FALSE,verbose=FALSE,
                   family="gaussian")
## store simulated data:
ysims[,i]<-ysim
## add current simulated data to data-frame:
dat$ysim<-ysim

## fit model to simulated data:
m<-lme4::lmer(ysim ~ so + (so || subject) + (so||item),
              data=dat,
              control=lmerControl(calc.derivs=FALSE))
## save t-value
tvals[i]<-summary(m)$coefficients[2,3]
}

mean(abs(tvals)>2)
```

```
## [1] 0.9
```

```
hist(gg05e1$rawRT,freq = FALSE,ylab="density",
```

```
     xlim=c(-1000,5000),
     main="",xlab="reading times (ms)")
for(i in 1:100){
  lines(density(ysims[,i]))
}
```



As the above plot shows, a model using raw readings times lead to predicted reading times including negative values, which is obviously implausible.

If we want to fit the model on log RTs, the code changes as follows. First, the model is fit to the data on the log ms scale:

```
m_lmer<-lmer(logrt~so + (1+so||subject)+(1+so||item),gg05e1)
summary(m_lmer)
```

```
## Linear mixed model fit by REML. t-tests use
##   Satterthwaite's method [lmerModLmerTest]
## Formula:
## logrt ~ so + (1 + so || subject) + (1 + so || item)
##    Data: gg05e1
##
```

```
## REML criterion at convergence: 704
##
## Scaled residuals:
##    Min     1Q Median     3Q    Max
## -2.960 -0.508 -0.131  0.278  6.125
##
## Random effects:
##  Groups    Name        Variance Std.Dev.
##  subject   (Intercept) 0.10090  0.3176
##  subject.1 so          0.04894  0.2212
##  item      (Intercept) 0.00127  0.0356
##  item.1    so          0.00647  0.0805
##  Residual              0.13063  0.3614
## Number of obs: 672, groups:  subject, 42; item, 16
##
## Fixed effects:
##             Estimate Std. Error      df t value
## (Intercept)   5.8831     0.0517 42.1801  113.72
## so            0.1240     0.0485 28.6962    2.56
##             Pr(>|t|)
## (Intercept)   <2e-16
## so             0.016
##
## Correlation of Fixed Effects:
##    (Intr)
## so 0.000
```

The design remains unchanged; only the for-loop changes: the `simLMM` function is passed through the `exp()` function, to exponentiate the data generated on the log scale.

```
tvals<-rep(NA,100)

for(i in 1:100){
## generate data:
dat$ysim <- exp(simLMM(formula = ~ 1 + so + (1 + so | subject) + (1+so|item)
                  data = dat,
```

```
                    LMM = m_lmer,
                    CP =  0.0,
                    empirical=FALSE,verbose=FALSE,
                    family="gaussian"))

## fit model to simulated data:
m<-lme4::lmer(log(ysim) ~ so + (so || subject) + (so||item),
              data=dat,
              control=lmerControl(calc.derivs=FALSE))
## save t-value
tvals[i]<-summary(m)$coefficients[2,3]
}

mean(abs(tvals)>2)
```

```
## [1] 0.75
```

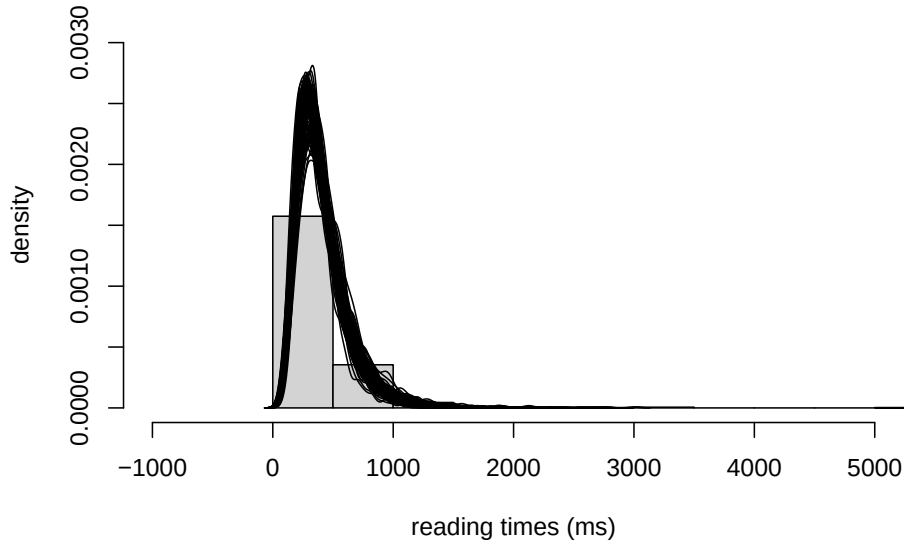
The model fit on log RTs produces much more realistic data:

```
n<-dim(dat)[1]
nsim<-100
ysims<-matrix(rep(NA,n*nsim),ncol=nsim)

for(i in 1:100){
## generate data:
ysim <- exp(simLMM(formula = ~ 1 + so + (1 + so | subject) +
                     (1+so|item),
                   data = dat,
                   LMM = m_lmer,
                   CP =  0.0,
                   empirical=FALSE,verbose=FALSE,
                   family="gaussian"))
## store simulated data:
ysims[,i]<-ysim
## add current simulated data to data-frame:
dat$ysim<-ysim
```

```
## fit model to simulated data:
m<-lme4::lmer(log(ysim) ~ so + (so || subject) + (so||item),
              data=dat,
              control=lmerControl(calc.derivs=FALSE))
## save t-value
tvals[i]<-summary(m)$coefficients[2,3]
}

mean(abs(tvals)>2)
```

```
## [1] 0.76
```

```
hist(gg05e1$rawRT,freq = FALSE,ylab="density",
     xlim=c(-1000,5000),
     main="",xlab="reading times (ms)",ylim=c(0,0.003))
for(i in 1:100){
  lines(density(ysims[,i]))
}
```



8.8.2 Simulating data in factorial designs

The `designr` library can be used to generate data from typical factorial designs used in psychology and linguistics.

8.9 Exercises

8.9.1 Drawing a power curve given a range of effect sizes

Use the simulation code as provided to compute a power function for effects sizes for the English relative clause effect ranging from 0.025, 0.05, 0.10, and 0.15, given that you have 16 items and 42 participants.

8.9.2 Power and log-transformation

Modify the simulation code to generate not log-normally distributed data, but normally distributed data. Refit the Grodner and Gibson (2005) data using raw reading times (i.e., do not log-transform them), and then use the parameter estimates from the data to compute a power function for effects sizes for the relative clause effect ranging from 10, 30, 60, 80 ms, given that you have 16 items and 42 participants. Compare your power curve with that of Part 1.

8.9.3 Evaluating models by generating simulated data

Generate data from the simulation function assuming a log-normal likelihood and then generate data from the function you wrote in Part 2 that assumes a normal likelihood. Compare the distributions of the two sets of simulated data to the observed distributions. Which simulation code produces more realistic data, and why?

8.9.4 Using simulation to check parameter recovery

Check whether the simulation code you wrote assuming a normal likelihood can recover the parameters.

8.9.5 Sample size calculations using simulation

Load the data-set shown below:

```
gibsonwu<-read.table("data/gibsonwucrit.txt")
```

Use simulation to determine how many subjects you would need to achieve power of 80%, given 16 items, and an effect size of 0.02 on the log ms scale. Draw a power curve: on the x-axis show the number of subjects, and on the y-axis the estimated power. Now draw two further curves, one for an effect size of 0.05 and another for an effect size of 0.10. This gives you a power curve, taking the uncertainty in the effect size into account.

```
library(knitr)
library(kableExtra)
library(tidyverse)
library(brms)
library(bcogsci)
library(papaja)
library(afex)
library(MASS)
library(hypr)
```

9

Understanding the multiple comparisons problem

to-do

```
library(knitr)
library(kableExtra)
library(tidyverse)
library(brms)
library(bcogsci)
library(papaja)
library(afex)
library(MASS)
library(hypr)
```

10

Model selection

to-do

Bibliography

Barr, D. J., Levy, R., Scheepers, C., and Tily, H. J. (2013). Random effects structure for confirmatory hypothesis testing: Keep it maximal. *Journal of Memory and Language*, 68(3):255–278.

Bates, D. M., Kliegl, R., Vasishth, S., and Baayen, H. (2015). Parsimonious mixed models. Unpublished manuscript.

Benjamin, D. J., Berger, J. O., Johannesson, M., Nosek, B. A., Wagenmakers, E.-J., Berk, R., Bollen, K. A., Brembs, B., Brown, L., Camerer, C., et al. (2018). Redefine statistical significance. *Nature Human Behaviour*, 2(1):6.

Blitzstein, J. K. and Hwang, J. (2014). *Introduction to probability*. Chapman and Hall/CRC.

Bolker, B. (2018). https://github.com/bbolker/mixedmodels-misc/blob/master/notes/contrasts.rmd.

Button, K. S., Ioannidis, J. P., Mokrysz, C., Nosek, B. A., Flint, J., Robinson, E. S., and Munafò, M. R. (2013). Power failure: why small sample size undermines the reliability of neuroscience. *Nature Reviews Neuroscience*, 14(5):365–376.

Cohen, J. (1988). *Statistical power analysis for the behavioral sciences*. Lawrence Erlbaum, Hillsdale, NJ, 2 edition.

Cumming, G. (2014). The new statistics: Why and how. *Psychological science*, 25(1):7–29.

Dobson, A. J. and Barnett, A. (2011). *An introduction to generalized linear models*. CRC press.

Fieller, N. (2016). *Basics of matrix algebra for statistics with R*. CRC Press, Boca Raton, FL.

Fox, J. (2009). *A mathematical primer for social statistics*. Number 159. Sage.

Frank, S. L., Trompenaars, T., and Vasishth, S. (2015). Cross-linguistic differences in processing double-embedded relative clauses: Working-memory constraints or language statistics? *Cognitive Science*, 40:554–578.

Friendly, M., Fox, J., and Chalmers, P. (2020). *matlib: Matrix Functions for Teaching and Learning Linear Algebra and Multivariate Statistics*. R package version 0.9.3.

Gelman, A., Carlin, J. B., Stern, H. S., Dunson, D. B., Vehtari, A., and Rubin, D. B. (2014). *Bayesian Data Analysis*. Chapman and Hall/CRC, Boca Raton, FL, third edition.

Gibson, E. and Thomas, J. (1999). Memory limitations and structural forgetting: The perception of complex ungrammatical sentences as grammatical. *Language and Cognitive Processes*, 14(3):225–248.

Gibson, E. and Wu, H.-H. I. (2013). Processing Chinese relative clauses in context. *Language and Cognitive Processes*, 28(1-2):125–155.

Gill, J. (2006). *Essential mathematics for political and social research*. Cambridge University Press Cambridge.

Grodner, D. and Gibson, E. (2005). Consequences of the serial nature of linguistic input. *Cognitive Science*, 29:261–290.

Hedges, L. V. (1984). Estimation of effect size under nonrandom sampling: The effects of censoring studies yielding statistically insignificant mean differences. *Journal of Educational Statistics*, 9(1):61–85.

Heister, J., Würzner, K.-M., and Kliegl, R. (2012). Analysing large datasets of eye movements during reading. *Visual word recognition*, 2:102–130.

Hoenig, J. M. and Heisey, D. M. (2001). The abuse of power: The

pervasive fallacy of power calculations for data analysis. *The American Statistician*, 55:19–24.

Hsiao, F. P.-F. and Gibson, E. (2003). Processing relative clauses in Chinese. *Cognition*, 90:3–27.

Husain, S., Vasishth, S., and Srinivasan, N. (2015). Integration and prediction difficulty in Hindi sentence comprehension: Evidence from an eye-tracking corpus. *Journal of Eye Movement Research*, 8(2):1–12.

Ioannidis, J. P. (2008). Why most discovered true associations are inflated. *Epidemiology*, 19(5):640–648.

Jäger, L. A., Engelmann, F., and Vasishth, S. (2017). Similarity-based interference in sentence comprehension: Literature review and Bayesian meta-analysis. *Journal of Memory and Language*, 94:316–339.

Johnson, K. (2011). *Quantitative methods in linguistics*. John Wiley & Sons.

Kerns, G. J. (2010). *Introduction to Probability and Statistics Using R*.

Lane, D. M. and Dunlap, W. P. (1978). Estimating effect size: Bias resulting from the significance criterion in editorial decisions. *British Journal of Mathematical and Statistical Psychology*, 31(2):107–112.

Laurinavichyute, A. (2020). *Similarity-based interference and faulty encoding accounts of sentence processing*. dissertation, University of Potsdam.

Matuschek, H., Kliegl, R., Vasishth, S., Baayen, R. H., and Bates, D. M. (2017). Balancing Type I Error and Power in Linear Mixed Models. *Journal of Memory and Language*, 94:305–315.

Miller, I. and Miller, M. (2004). *John E. Freund's Mathematical Statistics with Applications*. Prentice Hall.

Moore, W. H. and Siegel, D. A. (2013). *A mathematics course for political and social research*. Princeton University Press.

Morin, D. J. (2016). *Probability: For the Enthusiastic Beginner*. Createspace Independent Publishing Platform.

Nieuwenhuis, R., Te Grotenhuis, M., and Pelzer, B. (2012). influence.me: Tools for detecting influential data in mixed effects models. *R Journal*, 4(2):38–47.

Pocock, S. J. (2013). *Clinical trials: A practical approach*. John Wiley & Sons.

R Core Team (2019). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria.

Rabe, M. M., Vasishth, S., Hohenstein, S., Kliegl, R., and Schad, D. J. (2020). hypr: An r package for hypothesis-driven contrast coding. *Journal of Open Source Software*, 5(48):2134.

Rice, J. A. (1995). *Mathematical statistics and data analysis*. Duxbury press Belmont, CA.

Ripley, B. (2019). *MASS: Support Functions and Datasets for Venables and Ripley's MASS*. R package version 7.3-51.4.

Royall, R. (1997). *Statistical Evidence: A likelihood paradigm*. Chapman and Hall, CRC Press, New York.

Schad, D. J., Vasishth, S., Hohenstein, S., and Kliegl, R. (2020a). How to capitalize on a priori contrasts in linear (mixed) models: A tutorial. *Journal of Memory and Language*, 110.

Schad, D. J., Vasishth, S., Hohenstein, S., and Kliegl, R. (2020b). How to capitalize on a priori contrasts in linear (mixed) models: A tutorial. *Journal of Memory and Language*, 110:104038.

Snedecor, G. W. and Cochran, W. G. (1967). *Statistical Methods*. Iowa State University Press, Ames, Iowa.

Vasishth, S., Chen, Z., Li, Q., and Guo, G. (2013). Processing Chinese relative clauses: Evidence for the subject-relative advantage. *PLoS ONE*, 8(10):1–14.

Vasishth, S., Mertzen, D., Jäger, L. A., and Gelman, A. (2018).

The statistical significance filter leads to overoptimistic expectations of replicability. *Journal of Memory and Language*, 103:151–175.

Vasishth, S., Suckow, K., Lewis, R. L., and Kern, S. (2011). Short-term forgetting in sentence comprehension: Crosslinguistic evidence from head-final structures. *Language and Cognitive Processes*, 25:533–567.