Bruno Nicenboim, Daniel Schad, and Shravan Vasishth

An Introduction to Bayesian Data Analysis for Cognitive Science

This book is dedicated to the thousands, perhaps millions, of psycholinguists and psychologists struggling to understand what their data is/are trying to tell them.

Contents

List of Tables

List of Figures

Preface

This book is a relatively gentle introduction to carrying out Bayesian data analysis and cognitive modeling using the probabilistic programming language Stan (Carpenter et al., 2017), and the front-end to Stan called `brms` (Bürkner, 2019). Our target audience is cognitive scientists (e.g., linguists and psychologists) who carry out behavioral experiments, and who are interested in learning the Bayesian data analysis methodology from the ground up and in a principled manner. Our aim is to make Bayesian statistics a standard part of the data analysis toolkit for experimental linguistics, psycholinguistics, psychology, and related disciplines.

Many excellent introductory textbooks exist already for Bayesian data analysis. Why write yet another book? Our text is different from other attempts 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 unique complexities. We provide many examples of data-sets involving eyetracking (visual world and reading), self-paced reading, event-related potentials, reaction time, acceptability rating judgements, speeded grammaticality judgements, and question-response accuracies. Second, from the very outset, we stress a particular workflow that has as its centerpiece simulating data; we aim to teach a philosophy that involves thinking hard about the assumed underlying generative process, **even before the data are collected**. The data analysis approach that we hope to teach through this book involves a cycle of prior predictive and posterior predictive checks, and model validation using simulated data. We try to inculcate a sense of how inferences can be drawn from the posterior distribution of theoretically interesting parameters without resorting to binary decisions like "significant" or "not-significant". We are hopeful that this will set a new standard for reporting results of data analyses in a more nuanced manner, and lead to more measured claims in the published literature.

0.1 Prerequisites

Any rigorous introduction to Bayesian data analysis requires at least a passive knowledge of probability theory, calculus, and linear algebra. We do not require that the reader already has this background when they start the book. Instead, the relevant ideas are introduced informally just in time, as soon as they are needed. The reader is never required to have an active ability to solve probability problems, to solve integrals or compute derivatives, or to carry out matrix computations by hand. What we do expect is some relatively simple high school arithmetic and algebra; a quick look through chapter 1 of Gill (2006) before starting this book is highly recommended. We also expect that the reader is willing to learn enough of the programming language R (R Core Team, 2019) and Stan/brms to reproduce the examples presented. For newcomers to R, we provide a quick introduction in the appendix that covers all the constructs used in the book. There are many good online resources on R that the reader can consult. Examples are: R for data science[1], and Efficient R programming[2].

We also assume that the reader is familiar with basic frequentist data analysis methodology; in particular, the reader should know how to carry out one and two sample t-tests, both paired and unpaired, and should know how to interpret the t-score and p-value that are computed from such tests. We remind the reader of these basic ideas in chapter 1, but we don't use up too much space in this book on comparing frequentist and Bayesian methods. We do not try to convince the reader to use the Bayesian approach over the frequentist one; our goal is to focus on the *what* and the *how* of Bayesian data analysis, and not the *why*. Other books and articles discuss the latter aspect in detail; for example, Kruschke (2014) compares frequentist and Bayesian methods in detail.

provide comprehensive book recommendations

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

0.2 How to read this book

The chapters in this book are intended to be read in sequence, but during the first pass through the book, the reader should feel free to completely skip the sections marked with an asterisk. These sections provide a more formal development that will be useful when the reader transitions to more advanced textbooks like Gelman et al. (2014).

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

0.3 Online materials

The entire book, including all data and source code, is available online for free on `https://github.com/vasishth/Bayes_CogSci`. The solutions to exercises are provided there under the directory solutions.

to-do: provide solutions

0.4 Software needed

```
##
## Attaching package: 'dplyr'

## The following objects are masked from 'package:stats':
##
##     filter, lag

## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

Before you start, please install

- R[3] (and RStudio[4], or any other Integrated Development Environment that you prefer)
- The R package `rstan` (please pay close attention to the installation instructions!):
  - Instructions for Windows[5]
  - Instructions for Mac or Linux[6]
- The R packages `MASS`, `dplyr`, `tidyr`, `purrr`, `readr`, `extraDistr`, `ggplot2`, `brms`, `bayesplot`, `tictoc`, can be installed the usual way: `install.packages(c("MASS", "dplyr", "tidyr", "purrr", "readr", "extraDistr", "ggplot2", "brms", "bayesplot", "tictoc")`

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

```
set.seed(42)
library(MASS)
##be careful to load dplyr after MASS
library(dplyr)
library(tidyr)
library(purrr)
library(readr)
library(extraDistr)
library(ggplot2)
library(brms)
library(rstan)
## Save compiled models:
rstan_options(auto_write = TRUE)
## Parallelize the chains using all the cores:
options(mc.cores = parallel::detectCores())
library(bayesplot)
library(tictoc)
library(gridExtra)
```

[3] https://cran.r-project.org/
[4] https://www.rstudio.com/
[5] https://github.com/stan-dev/rstan/wiki/Installing-RStan-on-Windows
[6] https://github.com/stan-dev/rstan/wiki/Installing-RStan-on-Mac-or-Linux

```
# To solve some conflicts between  packages
select <- dplyr::select
extract <- rstan::extract
```

0.5 Acknowledgments

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). The participants in these courses helped us considerably in improving the material presented here. We are also grateful to members of Vasishth lab for comments on earlier drafts of this book.

Vasishth acknowledges the University of Potsdam for granting a sabbatical semester during 2019-20, and the Zentrum für Interdisziplinäre Forschung (ZiF) at the University of Bielefeld, Germany, for providing time for writing during Septemer 2019; this stay at ZiF was part of the activities of the research group *Statistical Models for Psychological and Linguistic Data* (led by Reinhold Kliegl, Douglas Bates, and Harald Baayen).

This book would have been impossible to make without the following software: R (Version 3.6.1; R Core Team, 2019) and the R-packages *DT* (Version 0.10; Xie et al., 2018), *MASS* (Version 7.3.51.4; Ripley, 2019), *Matrix* (Version 1.2.18; Bates and Maechler, 2019), *Rcpp* (Version 1.0.3; Eddelbuettel et al., 2019), *StanHeaders* (Version 2.19.0; Goodrich et al., 2019), *bayesplot* (Version 1.7.1; Gabry and Mahr, 2018), *bibtex* (Version 0.4.2; Francois, 2017), *bookdown* (Version 0.16; Xie, 2019a), *brms* (Version 2.10.0; Bürkner, 2019), *citr* (Version 0.3.2; Aust, 2019), *dplyr* (Version 0.8.3; Wickham et al., 2019b), *extraDistr* (Version 1.8.11; Wolodzko, 2019), *gdtools* (Version 0.2.1; Gohel et al., 2019), *ggplot2* (Version 3.2.1; Wickham et al., 2019a), *gridExtra* (Version 2.3; Auguie, 2017), *htmlwidgets* (Version 1.5.1; Vaidyanathan et al., 2018), *knitr* (Version 1.26; Xie, 2019b), *lme4* (Version 1.1.21; Bates

et al., 2015a), *miniUI* (Version 0.1.1.1; Cheng, 2018), *purrr* (Version 0.3.3; Henry and Wickham, 2019), *readr* (Version 1.3.1; Wickham et al., 2018), *rmarkdown* (Version 1.18; Allaire et al., 2019), *rstan* (Version 2.19.2; Guo et al., 2019), *servr* (Version 0.15; Xie, 2019c), *styler* (Version 1.2.0; Müller and Walthert, 2019), *tictoc* (Version 1.0; Izrailev, 2014), *tidyr* (Version 1.0.0; Wickham and Henry, 2019), and *webshot* (Version 0.5.2; Chang, 2018)

Shravan Vasishth, Bruno Nicenboim, Daniel Schad, Potsdam, Germany

About the Authors

Bruno Nicenboim (http://www.ling.uni-potsdam.de/~nicenboim/) is a postdoctoral researcher at the University of Potsdam, Germany. He started studying Electronic Engineering in the National University of Rosario, Argentina, then transitioned to Human Sciences and spent eight years in Israel where he completed a Bachelors degree in Sociology and Linguistics and Masters degree in Linguistics in Tel Aviv University. During this time, he also worked in several IT companies. He is currently in Germany, where he completed a PhD in Cognitive Science in the University of Potsdam. His research interests are sentence comprehension, memory processes, decision making, and predictions.

Daniel J. Schad (https://danielschad.github.io/) is a postdoctoral researcher at the University of 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 is currently at the University of Potsdam, Germany, working on quantitative methods in Cognitive Science, including contrasts, properties of significance tests, Bayesian Workflow, and Bayes factor analyses.

Shravan Vasishth (http://vasishth.github.io) is professor of Psycholinguistics at the University of Potsdam, Germany. He holds the chair Psycholinguistics and Neurolinguistics (Language Processing). After completing his Bachelors degree in Japanese from Jawaharlal Nehru Uni-

versity, 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 and a PhD in Linguistics from the Ohio State University, Columbus, USA, and an MSc in Statistics from the School of Mathematics and Statistics, University of Sheffield, UK. He is a professional member of the Royal Statistical Society (GradStat ID: 128307), and a member of the International Society for Bayesian Analysis. Shravan Vasishth is professor of linguistics at the University of Potsdam, Germany. 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. His research is in the area of computational modeling of human sentence comprehension processes. He runs an annual summer school, Statistical Methods in Linguistics and Psychology (SMLP): vasishth.github.io/smlp. He regularly teaches short (two days to one-/two-week long) courses on statistical data analysis (Bayesian and frequentist methods).

1

Introduction

The central idea we will explore in this book is: given some data, how to use Bayes' theorem to quantify uncertainty about our belief regarding a scientific question of interest. Before we get into the details of the underlying theory and its application, some familiarity with the following topics needs to be in place: the basic concepts behind probability, the concept of random variables, probability distributions, and the concept of likelihood. We therefore turn to these topics first.

1.1 Probability

Informally, we all understand what the term "probability" means. We routinely talk about things like the probability of it raining today. However, there are two distinct ways to think about probability. One can think of the probability of an event with reference to the frequency with which it might occur in repeated observations. Such a conception of probability is easy to imagine in cases where an event can, at least in principle, occur repeatedly. An example would be obtaining a 6 when tossing a die again and again. However, this frequentist view of probability is difficult to justify when talking about certain one-of-a-kind events, such as earthquakes. In such situations, probability is expressing our uncertainty about the event happening. Moreover, we could even be uncertain about exactly how probable the event in question is; for example, we might say something like "I am 90% certain that the probability of an earthquake happening in the next year is between 10 and 40%". In this book, we will be particularly interested in quantifying uncertainty in this way: we will always want to know how unsure we are of the estimate we are interested in.

Both the frequency-based and the uncertain-belief perspective have their place in statistical inference, and depending on the situation, we are going to rely on both ways of thinking. Regardless of these differences in perspective, the probability of an event happening is defined to be constrained in the following way.

- The probability of an event must lie between 0 and 1, where 0 means that the event is impossible and cannot happen, and 1 means that the event is certain to happen.
- For any two mutually exclusive events, the probability that one or the other occurs is the sum of their individual probabilities.
- Two events are independent if and only if the probability of both events happening is equal to the product of the probabilities of each event happening.
- The probabilities of all possible events in the entire sample space must sum up to 1.

The above defitions are based on the axiomatic definition of probability by Kolmogorov (2018).

In the context of data analysis, we will talk about probability in the following way. Consider some data that you might have collected. This could be discrete 0,1 responses in a question-response accuracy task, or continuous measurements of reading times in milliseconds from an eyetracking study, etc. In any such cases, we will say that the data are being generated from a so-called **random variable**, which we will designate with a capital letter such as Y.[1]

The actually observed data will be distinguished from the random variable that generated it by using lower case y. We can call y an instance of Y; every new set of data will be slightly different due to random variability.

So what is a random variable? As a concrete example, consider an experiment where we ask subjects to respond to 10 questions that can be can either have a correct or incorrect answer. We will say that the number

[1] Here, we use Y, but we could have used any letter, such as X, Z, …. Later on, in some situations we will use Greek letters like θ, μ, σ to represent a random variable; however, following onvention in statistics, we will always use lower-case Greek letters for these.

of correct responses from a subject are instances of a random variable Y. Because only discrete responses are possible (the number of correct responses can be 0, 1, 2, ..., 10), this is an example of a **discrete random variable**.

This random variable will be assumed to have a parameter θ that represents the probability of producing a correct response. In statistics, given some observed data, typically our goal is to obtain an estimate of this parameter's true (unknown) value.

We will follow the convention that the actually observed number of correct responses is written as y, as opposed to the abstract random variable Y. As mentioned above, given that we have 10 trials, y can have values 0,1,2,...,10.

This discrete random variable Y has associated with it a function called a **probability mass function** or PMF. This function, which is written $p(y)$, gives us the probability of obtaining each of these 11 possible correct responses. We will write that this PMF depends on, or is conditional on, a particular fixed but unknown value for θ; the PMF will be written $p(y|\theta)$.

In frequentist approaches to data analysis, the observed data y are used to draw inferences about θ. A typical question that we ask in the frequentist paradigm is: does θ have a particular value θ_0? One can obtain estimates of the unknown value of θ from the observed data y, and then draw inferences about how different–or more precisely how far away–this estimate is from the hypothesized θ_0. This is the essence of null hypothesis significance testing. The conclusions from such a procedure are framed in terms of either rejecting the hypothesis that θ has value θ_0, or failing to reject this hypothesis. Here, rejecting the null hypothesis is the primary goal of the statistical test.

Bayesian data analysis begins with a different question. What is common to the frequentist paradigm is the assumption that the data are generated from a random variable Y and that there is a function $p(y|\theta)$ indexed by the parameter θ. Where the Bayesian approach diverges from the frequentist one is that the goal now is to express our uncertainty about θ. In other words, we treat the parameter θ itself as a random variable, which means that we assign a probability distribution $p(\theta)$ to this random variable. This distribution $p(\theta)$ is called the **prior distribution** on θ; such a

distribution could express our belief about the probability of correct responses, before we observe any data.

In a later chapter, we will spend some time trying to understand how such a prior distribution can be defined for a range of different research problems.

Given such a prior distribution and some data y, the end-product of a Bayesian data analysis is the so-called **posterior distribution** of the parameter given the data: $p(\theta|y)$. This posterior distribution is the probability distribution of θ after conditioning on y, i.e., after the data has been observed and is therefore known. All our statistical inference is based on this posterior distribution of θ; we can even carry out hypothesis tests analogous to the frequentist one sketched above.

We already mentioned conditional probability above when discussing the probability of the data given some parameter θ, which we wrote as the PMF $p(y|\theta)$. Conditional probability is an important concept in Bayesian data analysis, not least because it allows us to derive Bayes' theorem. Let's look at the definition of conditional probability next.

1.2 Conditional probability

Suppose that A stands for some discrete event; an example would be "the streets are wet." Suppose also that B stands for some other discrete event; an example is "it has been raining." We can talk about the probability of the streets being wet given that it has rained; or more generally, the probability of A given that B has happened.

This kind of statement is written as $Prob(A|B)$ or more simply $P(A|B)$. This is the conditional probability of event A given B. Conditional probability is defined as follows.

$$P(A|B) = \frac{P(A,B)}{P(B)} \text{ where } P(B) > 0 \tag{1.1}$$

We can rearrange the above equation so that we can talk about the joint

probability of both events A and B happening. This joint probability can be computed by first taking $P(B)$, the probability that event B (it has been raining) happens, and multipling this by the probability that A happens conditional on B, i.e., the probability that the streets are wet given it has been raining. This multiplication will give us $P(A, B)$, the joint probability of A and B, i.e., that it has been raining and that the streets are wet. We will write the above description as: $P(A, B) = P(A|B)P(B)$

to-do: include venn diagram?

Now, since the probability A and B happening is the same as the probability of B and A happening, i.e., since $P(B, A) = P(A, B)$, we can equate the expansions of these two terms:

$$P(A, B) = P(A|B)P(B) \text{ and } P(B, A) = P(B|A)P(A) \tag{1.2}$$

Equating the two expansions, we get:

$$P(A|B)P(B) = P(B|A)P(A) \tag{1.3}$$

to-do: do we need to spell this out more?

Dividing both sides by $P(B)$:

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)} \tag{1.4}$$

The above statement is Bayes' rule, and is the basis for all the statistical inference we will do in this book. For now, this is all the probability theory we need to know!

The next sections expand on the idea of a random variable, the probability distributions associated with the random variable, what it means to specify a prior distribution on a parameter, and how the prior and data can be used to derive the posterior distribution of θ.

To make the discussion concrete, we will use an example of a discrete random variable, the Binomial. Aftere discussing this discrete random variable, we present anotheer example, this time involving a continuous random variable, the Normal random variable.

The Binomial and Normal cases serve as the canonical examples that we will need in the initial stages of this book. We will introduce other random variables as needed: the Uniform, Beta, Poisson, Gamma, and the Exponential, among others. The properties of all the distributions we will eventually need are summarized in the appendix.

1.3 Discrete random variables: An example using the Binomial distribution

Consider the following sentence:

"It's raining, I'm going to take the …."

Suppose that our research goal is to estimate the probability, call it θ, of the word "umbrella" appearing in this sentence, versus any other word. If the sentence is completed with the word "umbrella", we will refer to it as a success; any other completion will be referred to as a failure. This is an example of a Binomial random variable: there can be only two possible outcomes, a success or a failure, and there is some true unknown probability θ of success that we want to estimate.[2]

One way to empirically estimate this probability of success is to carry out a so-called cloze task. In a cloze task, subjects are asked to complete a fragment of the original sentence, such as "It's raining, I'm going to take the …". The predictability or cloze probability of "umbrella" is then calculated as the proportion of times that the target word "umbrella" was produced as an answer by subjects.

Assume for simplicity that 10 subjects are asked to complete the above

[2] Technically, each single trial that can result in either a success or failure is called a Bernoulli random variable—but this random variable is just a special case of the Binomial when the number of trials is 1.

sentence; each subject does this task only once. This gives us independent responses from 10 trials that are either coded a success ("umbrella" was produced) or as a failure (some other word was produced). We can sum up the number of sucesses to calculate how many of the 10 trials had "umbrella" as a response. For example, if 8 instances of "umbrella" are produced in 10 trials, we would estimate the cloze probability of producing "umbrella" would be $8/10$.

We can repeatedly generate simulated sequences of the number of successes in R (later on we will demonstrate how to generate such random sequences of simulated data). Here is a case where we run the same experiment 20 times (the sample size is 10 each time).

```
rbinom(10,n=20,prob=0.5)
```

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

The number of successes in each of the 20 simulated experiments above is being generated by a discrete random variable Y with a probability distribution $p(y|\theta)$ called the **Binomial distribution**.

For discrete random variables such as the Binomial, the probability distribution $p(y|\theta)$ 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: $y = 0, 1, 2, ..., 10$ successes. Which of these outcomes is most probable depends on the parameter θ in the Binomial distribution that represents the probability of success.

The left-hand side plot in Figure 1.1 shows an example of a Binomial PMF with 10 trials, with the parameter θ fixed at 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.

to-do bar or line graphs above, instead of points

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

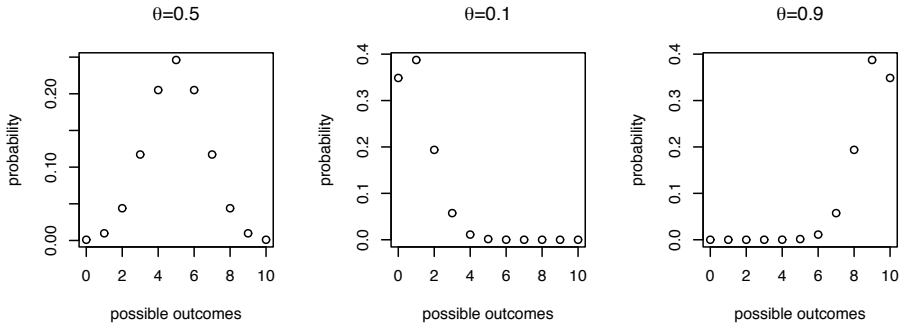


FIGURE 1.1: Probability mass functions of a binomial distribution assuming 10 trials, with 50%, 10%, and 90% probability of success.

$$\text{Binomial}(k|n,\theta) = \binom{n}{k}\theta^k(1-\theta)^{n-k} \tag{1.5}$$

Here, n represents the total number of trials, k the number of successes (this could range from 0 to 10), 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 secone trial could be a 1, etc. The term $\binom{n}{k}$ expands to $\frac{n!}{k!(n-k)!}$. In R, it is computed using the function `choose(n,k)`, with n and k representing positive integer values.

When we want to express the fact that the data is assumed to be generated from a Binomial random variable, we will write $Y \sim Binomial(n,\theta)$, where $\sim$ should be read as "is being generated from". If the data is generated from a random variable that has some other probability distribution $f(\theta)$, we will write $Y \sim f(\theta)$.

1.3.1 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. Without 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, n is a fixed number because we decide on the total number of trials before running the experiment. In the PMF θ is also a fixed value; the only variable in a PMF is k. 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) expectation E[Y]. Once we have estimated θ in this way, we can also obtain an estimate (also a maximum likelihood estimate) of the variance by computing $n\theta(1-\theta)$. These estimates are then used for statistical inference.

What does the term "maximum likelihood estimate" mean? The term **likelihood** refers to the Binomial distribution function, i.e., the PMF we saw above, $p(k|n,\theta)$. Recall that the PMF assumes that θ and n are fixed, and k will vary from 0 to 10 when the experiment is repeated multiple times. The likelihood function is the same function as the PMF, $p(k|n,\theta)$, but assumes that the data is fixed and only θ varies (from 0 to 1).

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:

$$\text{Binomial}(k = 7|n = 10, \theta) = \binom{10}{7}\theta^7(1-\theta)^{10-7} \tag{1.6}$$

Once we have observed the data (k=7 successes), 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:

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

Since the PMF and the likelihood refer to the same function seen in two different ways, sometimes the likelihood is written $p(\theta|k = 7, n =$

$10)$ to distinguish it from the PMF, which has the data appearing first ($p(k|n, \theta)$). We will write both the PMF and the likelihood identically in this book; context will disambiguate what we are referring to.

If we now plot the likelihood function for all possible values of θ ranging from 0 to 1, we get the plot shown in Figure 1.2.

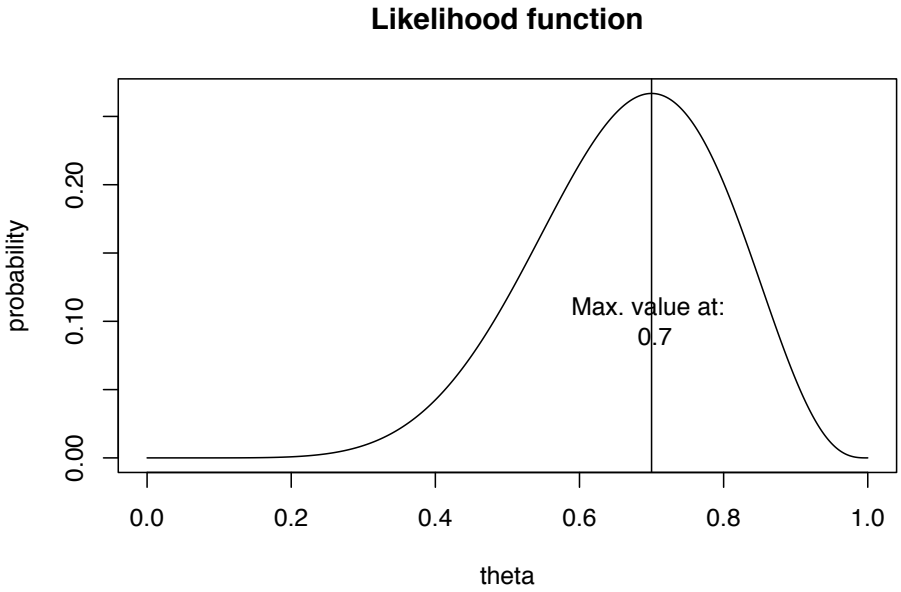


FIGURE 1.2: The likelihood function for 7 successes out of 10.

DS comment: do we want to show the code for computing all likelihood values? (maybe this comes later?)

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 that the parameter θ has, given the data.

It is crucial to note here that the phrase "most likely" here 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 as n approaches infinity.

1.3.2 What information does a probability distribution provide?

In Bayesian data analysis, we will constantly be asking the question: what information does a probability distribution give us? In particular, we will treat each parameter θ as a random variable; this will raise questions like: "what is the probability that the parameter θ lies between two values a and b;"what is the range over which we can be 95% certain that the true value of the parameter lies"? In order to be able to answer questions like these, we need to know what information we can obtain once we have a probability distribution, and how to extract this information. We therefore discuss the different kinds of information we can obtain from a probability distribution. For now we focus only on the Binomial random variable discussed above.

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

The Binomial distribution shown in Figure 1.1 already shows 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 the probability 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:

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

```
## [1] 0.25
```

The probabilities of success when θ is 0.1 or 0.9 can be computed by replacing 0.5 above by each of these probabilities. One can just do this by giving `dbinom` a vector of probabilities:

```
dbinom(5,size=10,prob=c(0.1,0.9))
```

```
## [1] 0.0015 0.0015
```

Note that the probability of a particular outcome is only computable in the discrete case; in the continuous case, this probability will always be zero (we discuss this when we turn to continuous probability distributions below).

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

Using the `dbinom` function, we can compute the cumulative probability of obtaining 1 or less, 2 or less successes etc. This is done 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] 0.055
```

Mathematically, we could write the above summation as:

$$\sum_{k=0}^{2} \binom{n}{k} \theta^k (1-\theta)^{n-k} \tag{1.7}$$

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] 0.055
```

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] 0.055
```

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] 0.95
```

The cumulative distribution function or CDF 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.

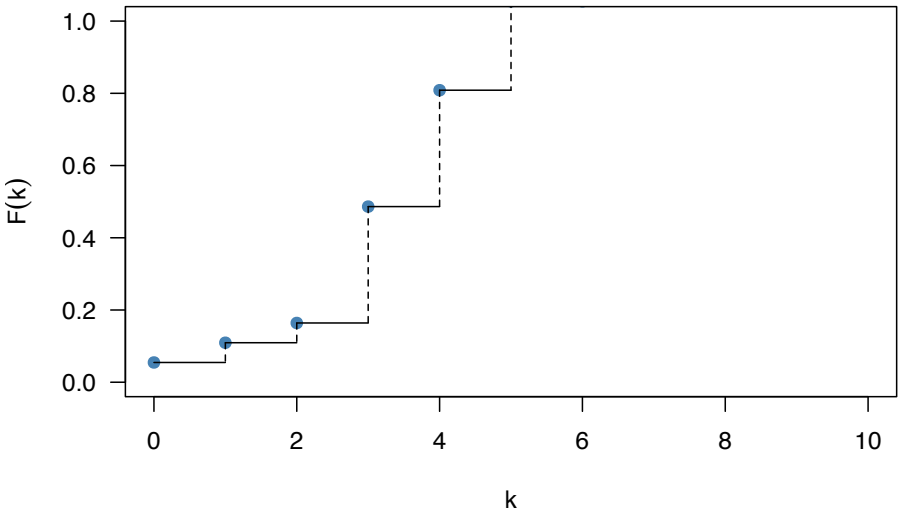


FIGURE 1.3: The cumulative distribution function for a Binomial distribution assuming 10 trials, with 50% probability of success.

1.3.2.3 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 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
```

to-do: explain why qbinom(0.77 gives 5 as an answer and not 4)

DS comment: maybe it's good to include an additional Figure for the inverse CDF and an example

1.3.2.4 Generate simulated data from a Binomial(n, θ) distribution

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

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

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

to-do: introduce Bernoulli here and link it with the code below

The above code generates the number of successes in an experiment with 10 trials. Repeatedly run the above code; you will get different sequences 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(10,size=1,prob=0.5)
mean(y)*10 ; sum(y)
```

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

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

1.4 Continuous random variables: An example using the Normal distribution

We will now revisit the idea of the random variable using a continuous distribution. Imagine that you have a vector of reading time data y measured in milliseconds and coming from a Normal distribution. The Normal distribution is defined in terms of two parameters: a mean value μ, which determines its center, and the variance σ^2, which determines how much spread there is around this center point.

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 some true, unknown mean, and σ^2 is some true, unknown variance of the Normal distribution that the reading times have been sampled from. There is a built-in function in R that computes the above function once we specify the mean μ and the standard deviation σ (in R, this parameter is specified in terms of the standard deviation rather than the variance).

Figure 1.4 visualizes the Normal distribution for particular values of μ and σ, as a PDF (using `dnorm`), a CDF (using `pnorm`), and the inverse CDF (using `qnorm`). It is clear from the figure that these are three different ways of looking at the same information.

to-do: Maybe this is the place to mention some interesting properties like: Normal(mu, sigma) = mu + Normal(0,1) * sigma (We'll use this property a lot later when we code in Stan)

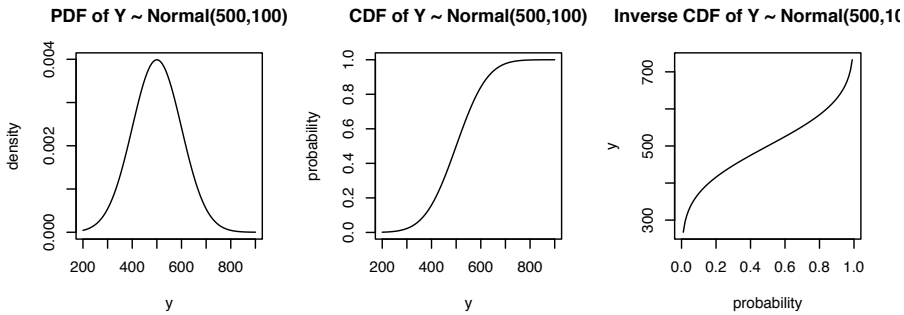


FIGURE 1.4: The PDF, CDF, and inverse CDF for the $Normal(\mu = 500, \sigma = 100)$.

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 a and b from a Normal distribution with mean μ and standard deviation σ?** Using the above example, we can ask what the probability of observing values between 200 and 700 ms:

```
pnorm(700,mean=500,sd=100)-pnorm(200,mean=500,sd=100)
```

```
## [1] 0.98
```

to-do: add figure illustrating the above

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 the area under the curve, and the area at any point on the x-axis is always 0. The implication here is that we can only ask about probabilities between two different points; e.g., the probability that Y lies between a and b, or $P(a < Y < b)$.

- **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 $P(Y < a)$.

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

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

The above output says that the probability that the random variable is less than $q = 695$ 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, the maximum likelihood estimates for the expectation and variance[3] are:

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

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

For example, we can 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] 560
```

```
## [1] 6778
```

Again, the sample mean and sample variance computed from a particular data-set need not necessarily be close to the true values of the respective parameters.

[3] R will compute variance by dividing by $n - 1$, not n; this is because dividing by n gives a biased estimate. This is not an important detail for our purposes, and in any case for large n it doesn't really matter whether one divides by n or $n - 1$.

1.4.1 An important distinction: probability vs. density in a continuous random variable

In continuous distributions like the Normal discussed above, it is important to understand that the probability density function or PDF, $p(y|\mu, \sigma)$ defines a mapping from the y values (the possible values that the data can have) to a quantity called the density of each possible value. We can see this function in action when we use `dnorm` to compute, say, the density value corresponding to $y = 1$ in the $Normal(\mu = 0, \sigma = 1)$ distribution.

```
## density:
dnorm(1,mean=0,sd=1)
```

```
## [1] 0.24
```

The quantity above is *not* the probability of observing 1 in this distribution. As mentioned earlier, probability in a continuous distribution is the area under the curve, and this area will always be zero at any point value. If we want to know the probability of obtaining values between an upper and lower bound b and a, i.e., $P(a < Y < b)$ where these are two distinct values, we must use the `pnorm` function. For example, the probability of observing a value between +2 and -2 in a Normal distribution with mean 0 and standard deviation 1 is:

```
pnorm(2,mean=0,sd=1)-pnorm(-2,mean=0,sd=1)
```

```
## [1] 0.95
```

Notice that the situation is different in discrete random variables. These have a probability mass function (PMF) associated with them—the Binomial distribution that we saw earlier is an example. There, the PMF maps the possible y values to the probabilities of those values occurring. That is why, in the Binomial distribution, the probability of observing exactly 2 successes when sampling from a $Binomial(n = 10, \theta = 0.5)$ can be computed using either `dbinom` or `pbinom`:

```
dbinom(2,size=10,prob=0.5)
```

```
## [1] 0.044
```

```
pbinom(2,size=10,prob=0.5)-pbinom(1,size=10,prob=0.5)
```

```
## [1] 0.044
```

In the second line of code above, we are computing the cumulative probability of observing two or less successes, minus the probability of observing one or less successes. This gives us the probability of observing exactly two successes. The `dbinom` gives us this same information.

1.5 An important concept: The marginal likelihood (integrating out a parameter)

Here, we introduce a concept that will turn up many times in this book. The concept we unpack here is called "integrating out a parameter". We will need this when we encounter Bayes' rule in the next chapter, and when we use Bayes factors later in the book.

Integrating out a parameter refers to the following situation. Suppose we have a Binomial random variable Y with PMF $p(Y)$. Suppose also that this PMF is defined in terms of parameter θ that can have only three possible values, $0.1, 0.5, 0.9$, each with equal probability. In other words, the probability that θ is $0.1, 0.5$, or 0.9 is 1/3 each.

We stick with our earlier example of $n = 10$ trials and $k = 8$ successes. The **likelihood function** then is

$$p(k = 8|n = 10, \theta) = \binom{10}{8}\theta^8(1-\theta)^2 \tag{1.11}$$

There is a related concept of **marginal likelihood**, which we can write here as $p(k = 8, n = 10)$. Marginal likelihood is the likelihood computed by "marginalizing" out the parameter θ: for each possible value that the

parameter θ can have, we compute the likelihood at that value and multiply that likelihood with the probability of that θ value occurring. Then we sum up each of the products computed in this way. Mathematically, this means that we carry out the following operation.

In our example, there are three possible values of θ, call them $\theta_1 = 0.1$, $\theta_2 = 0.5$, and $\theta_3 = 0.9$. Each has probability $1/3$; so $p(\theta_1) = p(\theta_2) = p(\theta_3) = 1/3$. Given this information, we can compute the marginal likelihood as follows:

$$\begin{aligned} p(k=8, n=10) = &\binom{10}{8}\theta_1^8(1-\theta_1)^2 \times p(\theta_1) \\ &+\binom{10}{8}\theta_2^8(1-\theta_2)^2 \times p(\theta_2) \\ &+\binom{10}{8}\theta_3^8(1-\theta_3)^2 \times p(\theta_3) \end{aligned} \tag{1.12}$$

Writing the θ values and their probabilities, we get:

$$\begin{aligned} p(k=8, n=10) = &\binom{10}{8}0.1^8(1-0.1)^2 \times \frac{1}{3} \\ &+\binom{10}{8}\theta_2^8(1-\theta_2)^2 \times \frac{1}{3} \\ &+\binom{10}{8}\theta_3^8(1-\theta_3)^2 \times \frac{1}{3} \end{aligned} \tag{1.13}$$

We can simplify this summation by collecting together the common terms:

$$\begin{aligned} p(k=8, n=10) = &\frac{1}{3}[\binom{10}{8}0.1^8(1-0.1)^2 \\ &+\binom{10}{8}\theta_2^8(1-\theta_2)^2 \\ &+\binom{10}{8}\theta_3^8(1-\theta_3)^2] \\ =&0.06 \end{aligned} \tag{1.14}$$

Thus, a marginal likelihood is a kind of weighted sum of the likelihood, weighted by the possible values of the parameter.[4]

The above example was contrived, because we stated that the parameter θ has only three possible values. In reality, because the parameter θ can have all possible values between 0 and 1, the summation has to be done over a continuous space $[0, 1]$. The way this summation is expressed in mathematics is through the integral symbol:

$$p(k = 8, n = 10) = \int_0^1 \binom{10}{8} \theta^8 (1-\theta)^2 \, d\theta \tag{1.15}$$

This statement is saying exactly what we computed above, except that the summation is being done over a continuous space ranging from 0 to 1. We say that the parameter θ has been integrated out, or marginalized. Integrating out a parameter will be a very common operation in this book, but fortunately we will never have to do the calculation ourselves. For the above case, we can easily compute the integral in R:

```
BinLik<-function(theta){
  choose(10,8)*theta^8 * (1-theta)^2
}
integrate(BinLik,lower=0,upper=1)$value
```

```
## [1] 0.091
```

This completes our discussion of random variables and probability distributions. We now summarize what we have learnt so far.

1.6 Summary of useful R functions relating to distributions

Table 1.1 summarizes the different functions relating to PMFs and PDFs, using the Binomial and Normal as examples.

[4] Where does the above formula come from? It falls out from the law of total probability; see Blitzstein and Hwang (2014) for a detailed exposition.

TABLE 1.1: Important R functions relating to random variables.

	Discrete	Continuous
Example:	Binomial$(y\|n,\theta)$	Normal$(y\|\mu,\sigma)$
Likelihood function	dbinom	dnorm
Prob Y=y	dbinom	always 0
Prob $Y \geq y, Y \leq y, y_1 < Y < y_2$	pbinom	pnorm
Inverse CDF	qbinom	qnorm
Generate simulated data	rbinom	rnorm

Later on, we will use other distributions, such as the Uniform, Beta, etc., and each of these has their own set of d-p-q-r functions in R. The appendix summarizes the properties of the distributions that we will need in this book.

1.7 Summary of concepts introduced in this chapter

to-do: add summary

1.8 Further reading

A quick review of the mathematical foundations needed for statistics is available in the short book by Fox (2009). Morin (2016) and Blitzstein and Hwang (2014) are accessible introductions to probability theory.

1.9 Exercises

1.9.1 Practice using the `pnorm` function

1.9.1.1 Part 1

Given a normal distribution with mean `61` and standard deviation `101`, use the pnorm function to calculate the probability of obtaining values between `217` and `-95` 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 `48` or less
- a score of `48` or more
- a score of `56` or more

1.9.1.3 Part 3

Given a normal distribution with mean `53` and standard deviation `4`, what is the probability of getting

- a score of `48` or less.
- a score between `50` and `56`.
- 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 `0.27`.
- the area (the probability) to the left of the upper boundary is `0.91`.

1.9.2.2 Part 2

Given a normal distribution with mean `56.93` and standard deviation `0.74`. There exist two quantiles, the lower quantile q1 and the upper quan-

tile q2, that are equidistant from the mean `56.93`, such that the area under the curve of the Normal probability between q1 and q2 is `85`%. Find q1 and q2.

1.9.3 Practice using `qt`

Take an independent random sample of size `144` from a normal distribution with mean `133`, and standard deviation `54`. 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 `130.63` and the sample standard deviation `50.05`.

- 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.

1.9.4 Maximum likelihood estimation 1

Given the data point `9.08`. The function `dnorm` gives the likelihood given a data point (or multiple data points) and a value for the mean and the standard deviation (sd). Using `dnorm`, compute

- the likelihood of the data point `9.08` assuming a mean of `12` and standard deviation 5.
- the likelihood of the data point `9.08` assuming a mean of `11` and standard deviation 5.
- the likelihood of the data point `9.08` assuming a mean of `10` and standard deviation 5.
- the likelihood of the data point `9.08` assuming a mean of `9` and standard deviation 5.

1.9.5 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] 504 503 497 487 507 506 492 484 502 497
```

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] 0.0017
```

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] 1.8
```

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

```
## [1] 1.8
```

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] -6.3
```

Even more compactly:

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

```
## [1] -6.3
```

- Given the 10 data points above, calculate the maximum likelihood estimate (MLE) of the expectation.
- The sum of the log-likelihoods of the data-points 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 `495.9`?
- Which value for the mean, the MLE or `495.9`, gives the higher log-likelihood?

2

Introduction to Bayesian data analysis

Recall Bayes' rule: When A and B are observable events, we can state the rule as follows:

$$P(A \mid B) = \frac{P(B \mid A)P(A)}{P(B)} \tag{2.1}$$

Given a vector of data y, Bayes' rule allows us to work out the posterior distributions of the parameters of interest, which we can repesent as the vector of parameters Θ. This computation is achieved by rewriting (2.1) as (2.2). What is different here is that Bayes' rule is written in terms of probability distributions. Here, $p(\cdot)$ is a probability density, not the probability of a single event, which we represent above using $P(\cdot)$.

$$p(\Theta|y) = \frac{p(y|\Theta) \cdot p(\Theta)}{p(y)} \tag{2.2}$$

The above statement can be rewritten in words as follows:

$$\text{Posterior} = \frac{\text{Likelihood} \cdot \text{Prior}}{\text{Marginal Likelihood}} \tag{2.3}$$

The terms here have the following meaning. We elaborate on each point with an example below.

- The *Posterior*, $p(\Theta|y)$ is the probability distribution of the parameters conditional on the data.
- The *Likelihood* is as described in chapter 1: it is the PMF (discrete case) or the PDF (continuous case) expressed a function of Θ.
- The *Prior* is the initial probability distribution of the parameter, before seeing the data.

- The *Marginal Likelihood* was introduced in chapter 1 and standardizes the posterior distribution to ensure that the area under the curve of the distribution sums to 1, that is, it ensures that the posterior is a valid probability distribution.

An example will clarify all these terms, as we explain below.

2.1 Deriving the posterior using Bayes' rule: An analytical example

Recall our cloze probability example earlier. Participants are shown sentences like

"It's raining. I'm going to take the …"

Ten participants are asked to complete the sentence. If 8 out of 10 participants complete the sentence with "umbrella," the estimated cloze probability or predictability (given the preceding context) would be $\frac{8}{10} = 0.8$. This is the maximum likelihood estimate of the probability of producing this word; we will designate the estimate with a "hat" on the parameter name: $\hat{\theta} = 0.8$.

Notice an important point here: one shortcoming of simply writing down the proportion in this way is that it ignores the uncertainty of our measurement: 0.8 could come from 10 participants ($\frac{8}{10}$), 100 participants ($\frac{80}{100}$), or $100,000$ participants ($\frac{80000}{100000}$). The uncertainty of the estimate 0.8 is different in each of these cases, and that is very relevant when drawing conclusions from data.

In the frequentist framework, the only thing we can characterize our uncertainty about is the **sampling distribution** of this parameter under imaginary repeated sampling; we can never talk about our uncertainty about the parameter's true value itself. Thus, for a sample size of 10, our uncertainty of the sampling distribution would be computed by calculating the sample variance σ^2 (here, $n \times \hat{\theta}(1-\hat{\theta}) = 10 \times 0.8 \times (1-0.8) = 1.6$), and then calculating the standard error: $\sigma/\sqrt{n} = 0.4$. Increasing the sample size will make this standard error smaller and smaller for the same estimated proportion of successes of 0.8. This increased precision

is a statement about the uncertainty of the sampling distribution of θ under imaginary repeated sampling; it is not an estimate of the uncertainty of θ itself.

The Bayesian framework gives us the opportunity to talk directly about our uncertainty of the parameter itself, given the data. This is achieved by obtaining the posterior distribution of the parameter using Bayes' rule, as we show below.

2.1.1 Choosing a likelihood

Under the assumptions we have set up above, the responses follow a Binomial distribution, and so the PMF can be written as follows.

$$p(k|n,\theta) = \binom{n}{k}\theta^k(1-\theta)^{n-k} \tag{2.4}$$

where k indicates the number of times "umbrella" is given as an answer, and n the total number of answers given.

In a particular experiment that we carry out, if $n = 10$ and $k = 8$, these data are now a fixed quantity. The PMF above now becomes a function of θ, the likelihood function:

$$p(k = 8|n = 10,\theta) = \binom{n}{k}\theta^8(1-\theta)^2 \tag{2.5}$$

The above function is a now a continuous function of the value θ, which has possible values ranging from 0 to 1. Compare this to the PMF of the Binomial, which treats θ as a fixed value and defines a discrete distribution over the n+1 possible discrete values k that we can observe (the possible number of successes).

It is important to pause for a moment here to appreciate the fact that the PMF and the likelihood are the same function seen from different points of view. The only difference between the two is what is considered to be fixed and what is varying. The PMF treats data as varying from experiment to experiment and θ as fixed, whereas the likelihood function treats the data as fixed and the parameter θ as varying.

We now turn our attention back to our main goal, which is to find out, using Bayes' rule, the posterior distribution of θ given our data: $p(\theta|n, k)$. In order to use Bayes' rule to calculate this posterior distribution, we need to define a prior distribution over the parameter θ. In doing so, we are explicitly expressing our prior uncertainty about plausible values of θ.

2.1.2 Choosing a prior for θ

For the choice of prior for θ in the Binomial distribution, we need to assume that the parameter θ is a random variable that has a PDF whose range lies within [0,1], the range over which θ can vary (this is because θ represents a probability). The Beta distribution, which is a PDF for a continuous random variable, is commonly used as prior for parameters representing probabilities. One reason for this choice is that its PDF rangers over the interval $[0, 1]$. The other reason for this choice is that it makes the Bayes' rule calculation remarkably easy.

The Beta distribution has the following PDF.

$$p(\theta|a, b) = \frac{1}{B(a, b)}\theta^{a-1}(1-\theta)^{b-1} \tag{2.6}$$

The term $B(a, b)$ expands to $\int_0^1 \theta^{a-1}(1-\theta)^{b-1}\, d\theta$, and is a normalizing constant rhat ensures that the area under the curve sums to one.[1]

The Beta distribution's parameters a and b can be interpreted as expressing our prior beliefs about the probability of success; a represents the number of "successes", in our case, answers that are "umbrella" and b the number of failures, the answers that are not "umbrella". Figure 2.1 shows the different Beta distribution shapes given different values of a and b.

As in the Binomial and Normal distributions that we saw in chapter 1, one

[1] In some textbooks, you may see the PDF of the Beta distribution with the normalizing constant $\frac{\Gamma(a+b)}{\Gamma(a)\Gamma(b)}$ (the expression $\Gamma(n)$ is defined as (n-1)!):

$$p(\theta|a, b) = \frac{\Gamma(a+b)}{\Gamma(a)\Gamma(b)}\theta^{a-1}(1-\theta)^{b-1}$$

These two statements for the Beta distribution are identical because $B(a, b)$ can be shown to be equal to $\frac{\Gamma(a)\Gamma(b)}{\Gamma(a+b)}$.

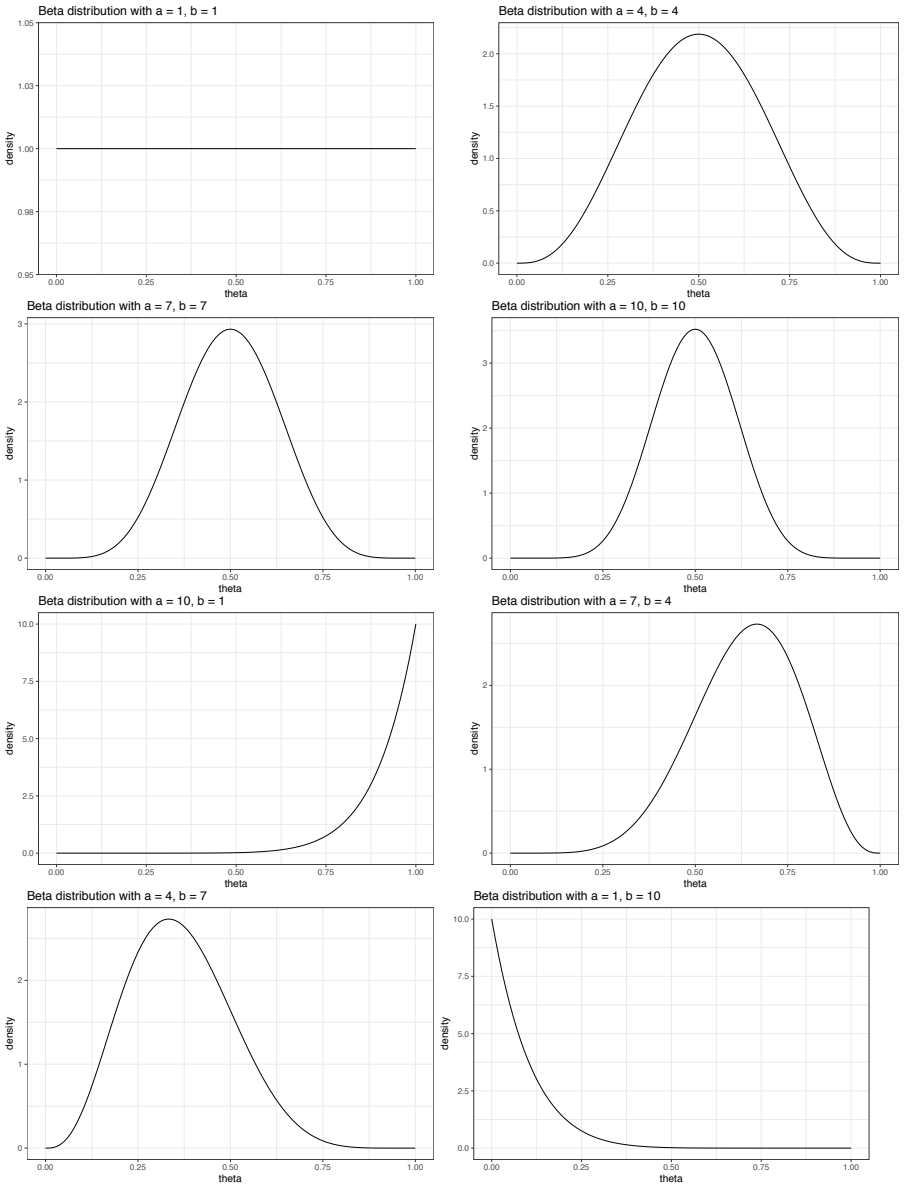


FIGURE 2.1: Examples of Beta distributions with different parameters.

can analytically derive the formulas for the expectation and variance of the Beta distribution. These are:

$$\mathrm{E}[X] = \frac{a}{a+b} \quad \mathrm{var}(X) = \frac{a \cdot b}{(a+b)^2(a+b+1)} \tag{2.7}$$

As an example, choosing $a = 4$ and $b = 4$ would mean that the answer "umbrella" is as likely as a different answer, but we are relatively unsure about this. We could express our uncertainty by computing the region over which we are 95% certain that the value of the parameter lies; this is the **95% credible interval**. For this, we would use the `qbeta` function in R; the parameters a and b are called `shape1` and `shape2` in `R`.

```
qbeta(c(0.025,0.975),shape1=4,shape2=4)
```

```
## [1] 0.18 0.82
```

If we were to choose $a = 10$ and $b = 10$, we would still be assuming that a priori the answer "umbrella" is just as likely as some other answer, but now our prior uncertainty about this mean is lower, as the 95% credible interval computed below shows.

```
qbeta(c(0.025,0.975),shape1=10,shape2=10)
```

```
## [1] 0.29 0.71
```

In Figure 2.1, we can see also the difference in uncertainty in these two examples graphically.

Which prior should we choose? In a real data analysis problem, the choice of prior would depend on what prior knowledge we want to bring into the analysis. If we don't have much prior information, we could use $a = b = 1$; this gives us a uniform prior. This kind of prior goes by various names: **non-informative prior**, or **weakly informative prior**. By contrast, if we have a lot of prior knowledge and/or a strong belief (e.g., based on a particular theory's predictions, or prior data) that θ has a particular range of plausible values, we can use a different set of a,b values to reflect our belief about the parameter. Notice in the above example that the larger

our parameters a and b, the narrower the spread of the distribution; i.e., the lower our uncertainty about the mean value of the parameter.

For the moment, just for illustration, we choose the values $a = 4$ and $b = 4$ for the Beta prior. Then, our prior for θ is the following Beta PDF:

$$p(\theta) = \frac{1}{B(4,4)}\theta^3(1-\theta)^3 \tag{2.8}$$

Having chosen a likelihood, and having defined a prior on θ, we are ready to carry out our first Bayesian analysis to derive a posterior distribution for θ.

2.1.3 Using Bayes' rule to compute the posterior $p(\theta|n,k)$

Having specified the likelihood and the prior, we will now use Bayes' rule to calculate $p(\theta|n,k)$. Using Bayes' rule simply involves replacing the Likelihood and the Prior we defined above into the equation we saw earlier:

$$\text{Posterior} = \frac{\text{Likelihood} \cdot \text{Prior}}{\text{Marginal Likelihood}} \tag{2.9}$$

Replace the terms for likelihood and prior into this equation:

$$p(\theta|n=10,k=8) = \frac{\left[\binom{10}{8}\theta^8 \cdot (1-\theta)^2\right] \times \left[\frac{1}{B(4,4)} \times \theta^3(1-\theta)^3\right]}{p(k=8)} \tag{2.10}$$

where $p(k=8)$ is $\int_0^1 p(k=8|n,\theta)p(\theta)\,d\theta$. This term will be a constant once the number of successes k is known; this is the marginal likelihood we encountered in chapter 1. In fact, once k is known, there are several constant values in the above equation; they are constants because none of them depend on the parameter of interest, θ. We can collect all of these together:

$$p(\theta|n=10,k=8) = \left[\frac{\binom{10}{8}}{B(4,4) \times p(k=8)}\right] [\theta^8(1-\theta)^2 \times \theta^3(1-\theta)^3] \tag{2.11}$$

The first term that is in square brackets, $\frac{\binom{10}{8}}{B(4,4)\times p(y)}$, is all the constants collected together, and is the normalizing constant we have seen before; it makes the posterior distribution $p(\theta|n = 10, k = 8)$ sum to one. Since it is a constant, we can ignore it for now and focus on the two other terms in the equation. Because we are ignoring the constant, we will now say that the posterior is proportional to the right-hand side.

to-do: introduce the idea of an unnormalized posterior here? see other suggestion elsewhere.

$$p(\theta|n = 10, k = 8) \propto [\theta^8(1-\theta)^2 \times \theta^3(1-\theta)^3] \tag{2.12}$$

A common way of writing the above equation is:

$$\text{Posterior} \propto \text{Likelihood} \times \text{Prior} \tag{2.13}$$

Resolving the right-hand side now simply involves adding up the exponents! In this example, computing the posterior really does boil down to this simple addition operation on the exponents.

$$p(\theta|n = 10, k = 8) \propto [\theta^{8+3}(1-\theta)^{2+3}] = \theta^{11}(1-\theta)^5 \tag{2.14}$$

The expression on the right-hand side corresponds to a Beta distribution with parameters $a = 12$, and $b = 6$. This becomes evidence if we rewrite the right-hand side such that it represents the core part of a Beta PDF. All that is missing is a normalizing constant which would make the area under the curve sum to one.

$$\theta^{11}(1-\theta)^5 = \theta^{12-1}(1-\theta)^{6-1} \tag{2.15}$$

This core part of any PDF or PMF is called the kernel of that distribution. Without a normalizing constant, the area under the curve will not sum to one. Let's check this:

```
PostFun<-function(theta){
  theta^11 * (1-theta)^5
}
(AUC<-integrate(PostFun,lower=0,upper=1)$value)
```

```
## [1] 0.000013
```

So the area under the curve (AUC) is not 1—the posterior that we computed above is not a proper probability distribution.

All that is needed to make this into a proper probability distribution is to include a normalizing constant, which, according to the definition of the Beta distribution, would be $B(12, 6)$. This term is in fact the integral we computed above.

$$p(\theta|n = 10, k = 8) = \frac{1}{B(12, 6)}\theta^{12-1}(1 - \theta)^{6-1} \tag{2.16}$$

Now, this function will sum to one:

```
PostFun<-function(theta){
  theta^11 * (1-theta)^5/AUC
}
round(integrate(PostFun,lower=0,upper=1)$value,2)
```

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

2.1.4 Summary of the procedure

To summarize, we started with a Binomial likelihood, multiplied it with the prior $\theta \sim Beta(4, 4)$, and obtained the posterior $p(\theta|n, k) \sim Beta(12, 6)$. The constants were ignored when carrying out the multiplication; we say that we computed the posterior **up to proportionality**. Finally, we showed how, in this simple example, the posterior can be rescaled to become a probability distribution, by including a proportionality constant.

The above example is a case of a **conjugate** analysis: the posterior on the

parameter has the same form as the prior. The above combination of likelihood and prior is called the Beta-Binomial conjugate case. There are several other such combinations of Likelihoods and Priors that yield a posterior that has the same PDF as the prior on the parameter; some examples will appear in the exercises.

Formally, conjugacy is defined as follows:

DEFINITION Given the likelihood $p(y|\theta)$, if the prior $p(\theta)$ results in a posterior $y(\theta|y)$ that has the same form as $p(\theta)$, then we call $p(\theta)$ a conjugate prior.

For the Beta-Binomial case, we can derive a very general relationship between the likelihood, prior, and posterior. Given the Binomial likelihood up to proportionality (ignoring the constant) $\theta^k(1-\theta)^{n-k}$, and given the prior, also up to proportionality, $\theta^{a-1}(1-\theta)^{b-1}$, their product will be:

$$\theta^k(1-\theta)^{n-k}\theta^{a-1}(1-\theta)^{b-1} = \theta^{a+k-1}(1-\theta)^{b+n-k-1} \tag{2.17}$$

Thus, given a $Binomial(n, k|\theta)$ likelihood, and a $Beta(a, b)$ prior on θ, the posterior will be $Beta(a + k, b + n - k)$.

2.1.5 Visualizing the prior, likelihood, and the posterior

We established in the example above that the posterior is a Beta distribution with parameters $a = 12$, and $b = 6$. We visualize the likelihood, prior, and the posterior alongside each other in 2.2.

We can summarize the posterior distribution either graphically as we did above, or summarize it by computing the mean and the variance. The mean gives us an estimate of the Cloze probability of producing "umbrella" in that sentence (given the model, i.e., given the likelihood and prior):

$$\mathrm{E}[\widehat{\theta}] = \frac{12}{12+6} = 0.67 \tag{2.18}$$

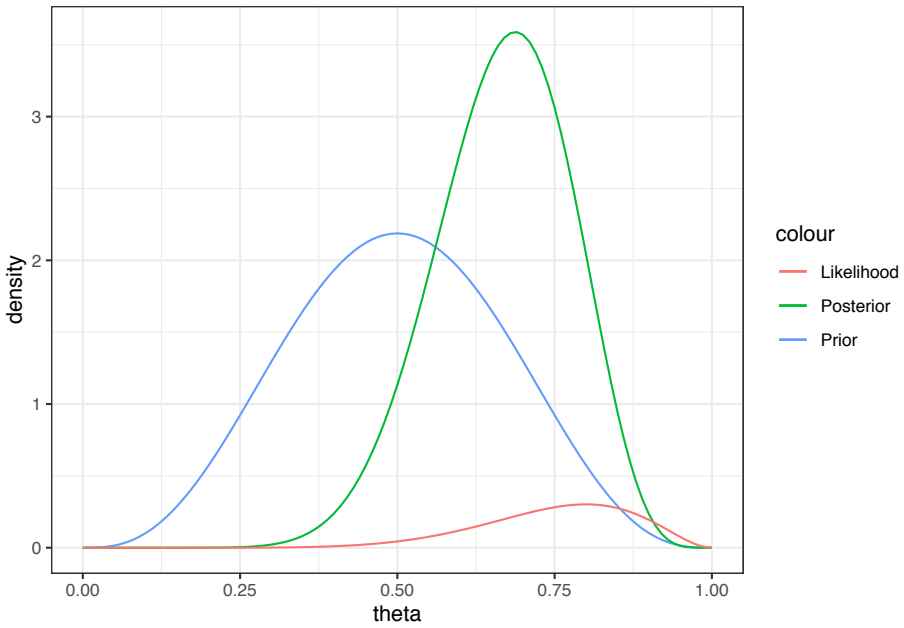


FIGURE 2.2: The likelihood, prior, and posterior in the Beta-Binomial example.

$$\text{var}[\hat{\theta}] = \frac{12 \cdot 6}{(12+6)^2(12+6+1)} = .01 \tag{2.19}$$

We could also display the 95% credible interval, the range over which we are 95% certain the true value of θ lies, given the data and model.

```
qbeta(c(0.025,0.975),shape1=12,shape2=6)
```

```
## [1] 0.44 0.86
```

Typically, we would summarize the results of a Bayesian analysis by displaying the posterior distribution of the parameter (or parameters) graphically, along with the above summary statistics: the mean, the standard deviation or variance, and the 95% credible interval. You will see many examples of such summaries later.

2.1.6 The posterior distribution is a compromise between the prior and the likelihood

Just for the sake of illustration, let's take four different Beta priors, each reflecting increasing certainty.

- Beta(a=2,b=2)
- Beta(a=3,b=3)
- Beta(a=6,b=6)
- Beta(a=21,b=21)

Each prior reflects a belief that $\theta = 0.5$, with varying degrees of (un)certainty. Given the general formula we developed above for the Beta-Binomial case, we just need to plug in the likelihood and the prior to get the posterior:

$$p(\theta|n,k) \propto p(k|n,\theta)p(\theta) \tag{2.20}$$

The four corresponding posterior distributios would be:

$$p(\theta \mid y,n) \propto [\theta^8(1-\theta)^2][\theta^{2-1}(1-\theta)^{2-1}] = \theta^{10-1}(1-\theta)^{4-1} \tag{2.21}$$

$$p(\theta \mid y,n) \propto [\theta^8(1-\theta)^2][\theta^{3-1}(1-\theta)^{3-1}] = \theta^{11-1}(1-\theta)^{5-1} \tag{2.22}$$

$$p(\theta \mid y,n) \propto [\theta^8(1-\theta)^2][\theta^{6-1}(1-\theta)^{6-1}] = \theta^{14-1}(1-\theta)^{8-1} \tag{2.23}$$

$$p(\theta \mid y,n) \propto [\theta^8(1-\theta)^2][\theta^{21-1}(1-\theta)^{21-1}] = \theta^{31-1}(1-\theta)^{23-1} \tag{2.24}$$

We can easily visualize each of these triplets of priors, likelihoods and posteriors. Use the Shiny app embedded below to visualize these different prior-likelihood combinations and look at the posterior in each case.

to-do: put in a shiny app that varies the a,b parameters and the amount of data, to show how the posterior is influenced by the data and the prior under different scenarios.

```
knitr::include_app("https://vasishth.shinyapps.io/AppTypeIPower",
  height = "500px")
```

```
## PhantomJS not found. You can install it with webshot::install_phantomjs(). If it is installed,
```

If you vary the prior's certainty (held constant at $n = 10, k = 8$ in the above example), the posterior orients itself increasingly towards the prior. In general, we can say the following about the likelihood-prior-posterior relationship:

- The posterior distribution is a compromise between the prior and the likelihood.
- For a given set of data, the greater the certainty in the prior, the more heavily the posterior will be influenced by the prior mean.
- Conversely, for a given set of data, the greater the **un**certainty in the prior, the more heavily the posterior will be influenced by the likelihood.

Another important observation emerges if we increase the sample size from 10 to, say, $1,000,000$. Suppose we still get a sample mean of 0.8 here, so that $k = 800,000$. Now, the posterior mean will be influenced almost entirely by the sample mean. This is because, in the general form for the posterior $Beta(a + k, b + n - k)$ that we computed above, the n and k become very large relative to the a, b values, and dominate in determining the posterior mean.

Whenever we do a Bayesian analysis, it is good practice to check whether the parameter you are interested in estimating is sensitive to the prior specification. Such an investigation is called a **sensitivity analysis**. Later in this book, we will see many examples of sensitivity analyses in realistic data-analysis settings.

2.1.7 Incremental knowledge gain using prior knowledge

In the above example, we used an artificial example where we asked 10 participants to complete the sentence shown at the beginning of the chapter, and then we counted the number of times that they produced "umbrella" vs. some other word as a continuation. Given 8 instances of "umbrella", and using a $Beta(4, 4)$ prior, we derived the posterior to be

$Beta(12, 6)$. We could now use this posterior as our prior for the next study. Suppose that we were to carry out a second experiment, again with 10 participants, and this time 6 produced "umbrella". We could now use our new prior (Beta(12,6)) to obtain an updated posterior. We have $a = 12, b = 6, n = 10, k = 6$. This gives us as posterior: $Beta(a + k, b + n - k) = Beta(12 + 6, 6 + 10 - 6) = Beta(18, 10)$.

Now, if we were to pool all our data from the 20 participants that we have now, then we would have had as data $n = 20, k = 14$. Suppose that we keep our initial prior of $a = 4, b = 4$. Then, our posterior would be $Beta(4 + 14, 4 + 20 - 14) = Beta(18, 10)$. This is exactly the same posterior that we got when first analyzed the first 10 participants' data, derived the posterior, and then used that posterior as a prior for the next 10 participants' data.

This toy example illustrates an important point that has great practical important for cognitive science. One can incrementally gain information about a research question by using information from previous studies and deriving a posterior, and then use that posterior as a prior. For practical examples from psycholinguistics showing how information can be pooled from previous studies, see Jäger et al. (2017) and Nicenboim et al. (2018). Vasishth and Engelmann (2020) illustrates an example of how the posterior from a previous study or collection of studies can be used to compute the posterior derived from new data. We return to this point in later chapters.

to-do: check that we do.

2.2 Summary of concepts introduced in this chapter

In this chapter, we learnt how to use Bayes' rule in the specific case of a Binomial likelihood, and a Beta prior on the θ parameter in the likelihood function. Our goal in any Bayesian analysis will follow the path we took in this simple example: decide on an appropriate likelihood function, decide on priors for all the parameters involved in the likelihood function,

and using this model (i.e., the likelihood and the priors) derive the posterior distribution of each parameter. Then we draw inferences about our research question based on the posterior distribution of the parameter.

In the example discussed in this chapter, Bayesian analysis was easy. This was because we considered the simple conjugate case of the Beta-Binomial. In realistic data-analysis settings, our likelihood function will be very complex, and many parameters will be involved. Multiplying the likelihood function and the priors will become mathematically difficult or impossible. For such situations, we use computational methods to obtain samples from the posterior distributions of the parameters.

to-do: add summary

2.3 Further reading

2.4 Exercises

2.4.1 Deriving Bayes' rule

Let A and B be two observable events. P(A) is the probability that A occurs, and P(B) is the probability that B occurs. $P(A|B)$ is the conditional probability that A occurs given that B has happened. $P(A, B)$ is the joint probability of A and B both occurring.

You are given the definition of conditional probability:

$$P(A|B) = \frac{P(A, B)}{P(B)} \text{ where } P(B) > 0 \tag{2.25}$$

Using the above definition, and using the fact that $P(A, B) = P(B, A)$ (i.e., the probability of A and B both occurring is the same as the probability of B and A both occurring), derive an expression for $P(B|A)$. Show the steps clearly in the derivation.

2.4.2 Conjugate forms 1

2.4.2.1 Computing the general form of a PDF for a posterior

Suppose you are given data k consisting of the number of successes, coming from a $Binomial(n, \theta)$ distribution. Example data are shown below, generated with probability of success $\theta = 0.5$, just for illustration:

```
## data:
k<-rbinom(n=1,size=10,prob=0.5)
k
```

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

Here, n represents the number of trials, and k the number of successes. The above code and output is just an example, and is no longer relevant for the question below.

Given k successes in n trials coming from a Binomial distribution, we define a $Beta(a, b)$ prior on the parameter θ.

Write down the Beta distribution that represents the posterior, in terms of $a, b, n, andk$.

2.4.2.2 Practical application

We ask 10 yes/no questions from a participant, and the participant returns 0 correct answers. We assume a Binomial likelihood function for these data. Also assume a Beta(1,1) prior on the parameter θ, which represents the probability of success. Use the result you derived above to write down the posterior distribution of the θ parameter.

2.4.3 Conjugate forms 2

Suppose you have n independent and identically distributed data points from a distribution that has the likelihood function $f(x|\theta) = \theta(1 - \theta)^{\sum_{i=1}^{n} x_i}$, where the data points x can have values 0,1,2,.... Let the prior on θ be Beta(a,b), a Beta distribution with parameters a,b. The posterior distribution is a Beta distribution with parameters a* and b*. Determine these parameters in terms of a, b, and $\sum_{i=1}^{n} x_i$.

2.4.4 Conjugate forms 3

The Gamma distribution is defined in terms of the parameters a, b: Ga(a,b). The probability density function is:

$$Ga(a,b) = \frac{b^a \lambda^{a-1} \exp\{-b\lambda\}}{\Gamma(a)} \tag{2.26}$$

We have data $x_1, \ldots, x_n$, with sample size n that is exponentially distributed. The exponential likelihood function is:

$$p(x_1, \ldots, x_n|\lambda) = \lambda^n \exp\{-\lambda \sum_{i=1}^{n} x_i\} \tag{2.27}$$

It turns out that if we assume a Ga(a,b) prior distribution and the above likelihood, the posterior distribution is a Gamma distribution. Find the parameters a' and b' of the posterior distribution.

2.4.5 Conjugate forms 4

2.4.5.1 a. Computing the posterior

This is a contrived example. Suppose we are modeling the number of times that a speaker says the word "I" per day. This could be of interest if we are studying, for example, how self-oriented a speaker is. The number of times x that the word is uttered in over a oarticular time period (here, one day) can be modeled by a Poisson distribution:

$$f(x \mid \theta) = \frac{\exp(-\theta)\theta^x}{x!} \tag{2.28}$$

where the rate θ is unknown, and the numbers of utterances of the target word on each day are independent given θ.

We are told that the prior mean of θ is 100 and prior variance for θ is 225. This information is based on the results of previous studies on the topic. We will use the Gamma(a,b) density (see previous question) as a prior for θ because this is a conjugate prior to the Poisson distribution.

- First, visualize the prior. a Gamma density prior for θ based on the above information.

[Hint: Note that we know that for a Gamma density with parameters a, b, the mean is $\frac{a}{b}$ and the variance is $\frac{a}{b^2}$. Since we are given values for the mean and variance, we can solve for a,b, which gives us the Gamma density.]

```
x<-0:200
plot(x,dgamma(x,10000/225,100/225),type="l",lty=1,
     main="Gamma prior",ylab="density",
     cex.lab=2,cex.main=2,cex.axis=2)
```

- Next, derive the posterior distribution of the parameter θ up to proportionality, and write down the posterior distribution in terms of the parameters of a Gamma distribution.

2.4.5.2 b. Practical application

Suppose we know that the number of "I" utterances from a particular individual is $115, 97, 79, 131$. Use the result you derived above to obtain the posterior distribution. In other words, write down the a,b parameters of the Gamma distribution representing the posterior distribution of θ.

Plot the prior, likelihood, and the posterior alongside each other.

Now suppose you get one new data point: 200. Write down the updated posterior (the a,b parameters of the Gamma distribution) given this new data-point. Add the updated posterior to the plot you made above.

3

Computational Bayesian data analysis

In the previous chapter, we learned how to analytically derive the posterior distribution of the parameters in our model. In practice, however, this is possible for only a very limited number of cases. Although the numerator of the Bayes rule, the unnormalized posterior, is easy to calculate (by multiplying the probability density/mass functions analytically), the denominator, the marginal likelihood, requires us to integrate the numerator; see (3.1).

$$
\begin{aligned}
p(\Theta|y) &= \frac{p(y|\Theta) \cdot p(\Theta)}{p(y)} \\
p(\Theta|y) &= \frac{p(y|\Theta) \cdot p(\Theta)}{\int_{\Theta} p(y|\Theta) \cdot p(\Theta) d\Theta}
\end{aligned}
\tag{3.1}
$$

Unless we are dealing with conjugate distributions, the solution will be extremely hard to derive or there will be no analytical solution. This was the major bottleneck of Bayesian analysis in the past, and required Bayesian practitioners to program an approximation method by themselves before they could even begin the Bayesian analysis. Fortunately, many of the probabilistic programming languages freely available today (see the next section for a listing) allow us to define our models without having to acquire expert knowledge about the relevant numerical techniques.

3.1 Deriving the posterior through sampling

Let's say that we want to derive the posterior of the model from 2.1, that is, the posterior distribution of the Cloze probability of *"umbrella"*, θ, given the following data: a word (e.g., *"umbrella"*) was answered 80 out of 100

times, and assuming a binomial distribution as the likelihood function, and $Beta(a = 4, b = 4)$ as a prior distribution for the Cloze probability. If we have samples from the posterior distribution of θ, instead of an analytically derived posterior distribution, given enough samples we will have a good approximation of the real posterior distribution. Getting samples from the posterior will be the only viable option in the models that we will discuss in this book. By "getting samples", we are talking about a situation analogous to when we use `rbinom` or `rnorm` to obtain samples from a particular distribution.

Thanks to probabilistic programming languages, it will be relatively straightforward to get these samples, and we will discuss how we will do it in more detail in the next section. But for now let's assume that we used Stan, `brms`, or even our hand-made sampler, and we got 20000 samples from the posterior distribution of the Cloze probability, θ: 0.793, 0.776, 0.726, 0.685, 0.776, 0.801, 0.722, 0.78, 0.739, 0.802, 0.809, 0.736, 0.78, 0.793, 0.835, 0.791, 0.773, 0.744, 0.816, 0.807, ... Figure 3.1 shows that the approximation of the posterior looks quite similar to the real posterior. And in fact the difference between the true and the approximated mean and variance are -0.0001 and -0.00002 respectively.

3.1.1 Bayesian Regression Models using 'Stan': brms

The surge in popularity of Bayesian statistics is closely tied to the increase in computing power and the appearance of probabilistic programming languages, such as WinBUGS (Lunn et al., 2000), JAGS (Plummer, 2016), and more recently pymc3 (Salvatier et al., 2016) and Stan (Carpenter et al., 2017). These statistical languages allow the user to define models without having to deal (for the most part) with the complexities of the sampling process. However, they require learning a new language since they the user has to fully specify the statistical model using a particular syntax.[1] Furthermore, some knowledge of the sampling process is needed to correctly parameterize the models and to avoid convergences issues (these topics will be covered in detail later in this book).

There are some alternatives that allow Bayesian inference in `R` without having to fully specify the model "by hand". The packages `rstanarm`

[1] The python package pymc3 is an exception since it is fully integrated into python.

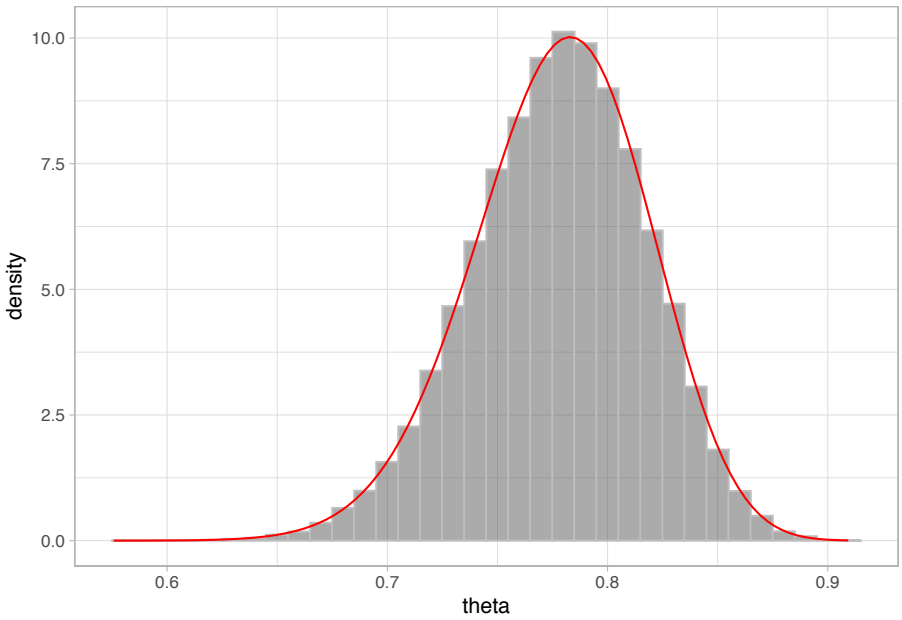


FIGURE 3.1: Histogram of the samples of θ from the posterior distribution calculated through sampling in gray; density plot of the exact posterior in red.

(Goodrich et al., 2018) and `brms` (Bürkner, 2019) provide Bayesian equivalents of many popular R model-fitting functions, such as (g)lmer (Bates et al., 2015b); both these packages use Stan for the back-end estimation and sampling.
JASP (JASP Team, 2019) provides a graphical user interface for both frequentist and Bayesian modeling, and is intended to be an open-source alternative to SPSS.

We will focus on `brms` in the first part of the book. This is because it can be useful for a smooth transition from frequentist models to their Bayesian equivalents. Although `brms` is powerful enough to satisfy the statistical needs of many cognitive scientists, it has the added benefit that the Stan code can be inspected (with the functions `make_stancode` and `make_standata`), allowing the users to customize their models or learn from the code produced internally by `brms` to eventually transition to write the models entirely in Stan.

3.1.1.1 A simple linear model: A single participant pressing a button repeatedly

We'll use the following example to illustrate the basic steps for fitting a model. Let's say we have data from a participant repeatedly pressing the space bar as fast as possible, without paying attention to any stimuli. The data are reaction times in milliseconds for each word in each trial. We would like to know how long it takes to press a key when there is no decision involved.

Let's model the data with the following assumptions:

1. There is a true underlying time, μ, that the participant needs to press the space bar.
2. There is some noise in this process.
3. The noise is normally distributed (this assumption is questionable given that reaction times are generally skewed; we fix this assumption later).

This means that the likelihood for each observation n will be:

$$rt_n \sim Normal(\mu, \sigma) \tag{3.2}$$

where $n = 1 \ldots N$, and rt is the dependent variable (reaction times in milliseconds). The variable N indexes the total number of data points. The letter μ indicates the *location* of the normal distribution function, the location parameter shifts the distribution left or right on the horizontal axis. For the normal distribution, the location is also the mean of the distribution. The letter σ indicates the *scale* of the distribution, as the scale decreases, the distribution gets narrower. The compressing approaches a spike (all the probability mass in one point) as the scale parameter goes to zero. For the normal distribution, the scale is also its standard deviation.

For a frequentist model that will give us the maximum likelihood estimate (the sample mean) of the time it takes to press the space bar, this would be enough information to write the formula in `R`, `rt ~ 1`, and plug it into the function `lm()` together with the data: `lm(rt ~ 1, data)`. The meaning of the `1` here is that there is no predictor associated with this parameter, and `lm` will estimate the so-called intercept of the model, in our case μ.

For a Bayesian model, we will also need to define priors for the two parameters of our model. Let's say that we know for sure that the time it takes to press a key will be positive and lower than a minute (60000ms), but we don't want to make a commitment regarding which values are more likely. We encode what we know about the noise in the task in σ: we know that this parameter must be positive and we'll assume that any value below 2000ms is equally likely. These priors are in general strongly discouraged because even when we know very little, a flat (or very wide) prior will almost never be the best approximation of what we know. We'll use them in this section for pedagogical purposes; the next chapter will show more realistic uses of priors.

$$\begin{aligned} \mu &\sim Uniform(0, 60000) \\ \sigma &\sim Uniform(0, 2000) \end{aligned} \tag{3.3}$$

We'll first load the data from `data/button_press.csv`:

```
df_noreading_data <- read_csv("./data/button_press.csv")
df_noreading_data
```

```
## # A tibble: 361 x 2
##      rt trialn
##   <dbl>  <dbl>
## 1   141      1
## 2   138      2
## 3   128      3
## 4   132      4
## 5   126      5
## # ... with 356 more rows
```

It is a good idea to look at the distribution of the data before doing anything else; see Figure 3.2. As we suspected, the data look a bit skewed, but we ignore this for the moment.

```
ggplot(df_noreading_data, aes(rt)) +
  geom_density() +
  ggtitle("Button-press data")
```

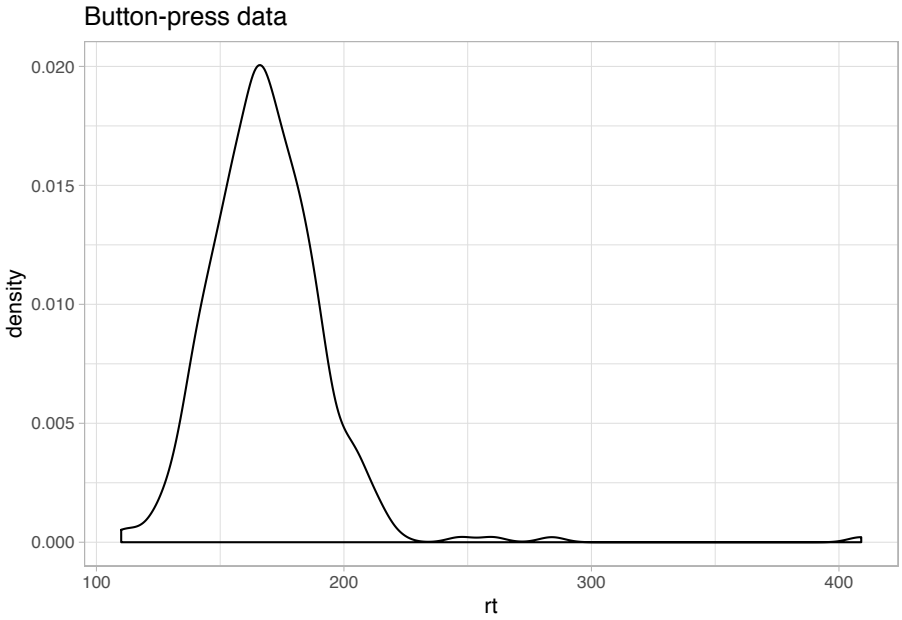


FIGURE 3.2: Visualizing the data

3.1.1.1.1 *Specifying the model in* `brms`

We'll fit the model defined by equations (3.2)-(3.2) with `brms` in the following way:

```
fit_press <- brm(rt ~ 1,
  data = df_noreading_data,
  family = gaussian(),
  prior = c(
    prior(uniform(0, 60000), class = Intercept),
    prior(uniform(0, 2000), class = sigma)
  ),
  chains = 4,
  iter = 2000,
  warmup = 1000
)
```

The `brms` code has some differences from a model fit with `lm` (or `lmer` from the `lme4` package). At this beginning stage, we'll focus on the following options:

1. The term `family = gaussian()` makes explicit that the underlying likelihood function is a normal distribution (Gaussian and normal are synonyms) that is implicit in lm(er). Other linking functions are possible, exactly as in the glm(er) function.
2. The term `prior` takes as argument a vector of priors. Although this specification of priors is optional, the researcher should always explicitly specify each prior. Otherwise, `brms` will define a prior by default, which may or may not be appropriate for the research area.
3. The term `chains` refers to the number of independent runs for sampling (by default four).
4. The term `iter` refers to the number of iterations that the sampler makes to sample from the posterior distribution of each parameter (by default 2000).
5. The term `warmup` refers to the number of iterations from the start of sampling that are eventually discarded (by default half of iter).

The last three options (together with `control` that was not used before) determine the behavior of the sampler algorithm: the No-U-Turn Sampler (NUTS; Hoffman and Gelman, 2014) extension of Hamiltonian Monte Carlo (Duane et al., 1987; Neal, 2011). We will discuss sampling in more depth in chapter ??, but we explain here the basic process.

3.1.1.1.2 Sampling and convergence in a nutshell

We start four chains independent from each other. Each chain "searches" for samples of the posterior in a multidimensional space, where each parameter corresponds to a dimension, and the shape of this space is determined by the priors and the likelihood. The chains start in random locations and in each iteration they take one sample each. The samples at the beginning do not belong to the posterior distribution. Eventually, the chains end up in the vicinity of the posterior distribution, and from

that point onwards the samples will belong to the posterior. That means that at the beginning the samples from the different chains will be far from each other, but that *at some point* they will converge. While there are no guarantees that we are running the chains for enough iterations, the default values of `brms` (and Stan) are in many cases enough to achieve that, and when they are not, we will receive warnings with recommendations. If the chains converged to the same distribution, by removing the "warmup" (also called burn-in) samples–by default half of a total of 2000 iterations–, we make sure that we do not get samples from the path to the posterior distribution; see figure 3.3. Stan runs diagnostics with the information from the chains, and if there are no warnings after fitting the model, we can be reasonable sure that the model converged and our samples are from the true posterior distribution. However, we do need to run more than one chain (preferably four), with a couple of thousands of iterations (at least) so that the diagnostics will work.

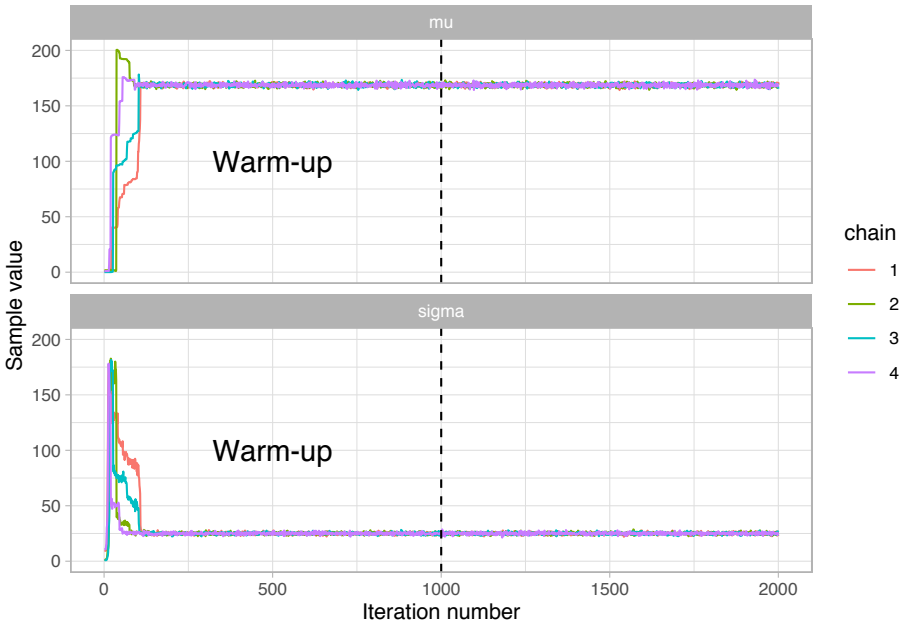


FIGURE 3.3: Trace plot of the `brms` model

3.1.1.1.3 *Output of* `brms`

If the model converged (i.e., if we didn't have any warning messages), the output of the sampling process shows the samples of the posterior distributions of each of the parameters:

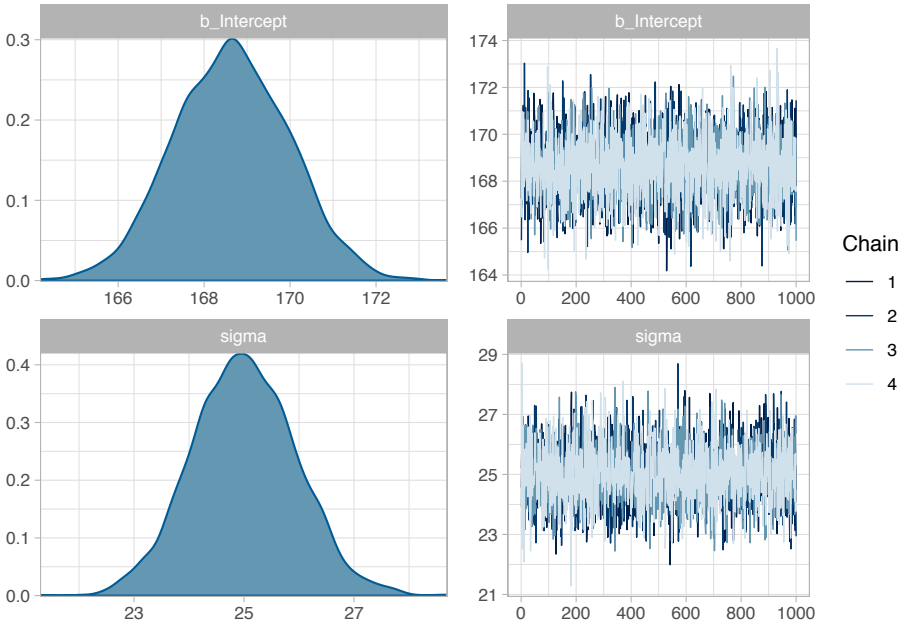
```
posterior_samples(fit_press) %>% str()
```

```
## 'data.frame':    4000 obs. of  3 variables:
##  $ b_Intercept: num  167 170 169 167 167 ...
##  $ sigma      : num  24.6 25.2 25.5 23.4 26.9 ...
##  $ lp__       : num  -1688 -1689 -1688 -1690 -1690 ...
```

Notice that `b_Intercept` corresponds to our μ and that `lp` is not really part of the posterior, it's the density of the unnormalized posterior for each iteration.

We can plot the histogram and the trace plot after the warmup:

```
plot(fit_press)
```



And `brms` provides a nice summary:

```
fit_press
# posterior_summary(fit_press) is also useful
```

```
##  Family: gaussian
##   Links: mu = identity; sigma = identity
## Formula: rt ~ 1
##    Data: df_noreading_data (Number of observations: 361)
## Samples: 4 chains, each with iter = 2000; warmup = 1000; thin = 1;
##          total post-warmup samples = 4000
##
## Population-Level Effects:
##           Estimate Est.Error l-95% CI u-95% CI Rhat
## Intercept   168.64      1.34   166.08   171.27 1.00
##           Bulk_ESS Tail_ESS
## Intercept     3648     2744
##
## Family Specific Parameters:
##       Estimate Est.Error l-95% CI u-95% CI Rhat
## sigma    25.01      0.94    23.17    26.90 1.00
##       Bulk_ESS Tail_ESS
## sigma     3311     2528
##
## Samples were drawn using sampling(NUTS). For each parameter, Eff.Sample
## is a crude measure of effective sample size, and Rhat is the potential
## scale reduction factor on split chains (at convergence, Rhat = 1).
```

Notice that the `Estimate` is just the mean of the posterior sample, and CI are the 95% quantiles:

```
posterior_samples(fit_press)$b_Intercept %>% mean()
```

```
## [1] 169
```

```
posterior_samples(fit_press)$b_Intercept %>% quantile(c(0.025, .975))
```

```
## 2.5%  98%
```

```
##  166  171
```

We see that we can fit our model without problems, and we get some posterior distributions for our parameters. However, we should ask ourselves the following questions:

1. What information are the priors encoding? Do the priors make sense?
2. Does the likelihood assumed in the model make sense for the data?

We'll try to answer these questions by looking at the *Prior and posterior predictive distributions*, and by doing sensitivity analyses as described in the following sections.

3.2 Prior predictive distribution

We had defined the following priors for our linear model:

$$\begin{aligned} \mu &\sim Uniform(0, 60000) \\ \sigma &\sim Uniform(0, 2000) \end{aligned} \tag{3.4}$$

These priors encode assumptions about the kind of data we would expect to see in a future study. To understand these assumptions, we are going to generate data from the model; such data, which is generated entirely by the prior distributions, is called the prior predictive distribution. Generating prior predictive distributions repeatedly helps us to check whether the priors make sense. What we want to know here is, do the priors generate realistic-looking data?

Formally, we want to know the density $p(\cdot)$ of data points $y_{pred_1,\ldots,y_pred_N}$ from a dataset D_{pred} of length N, given a vector of priors Θ and our likelihood $p(\cdot|\Theta)$; (in our example, $\Theta = \langle \mu, \sigma \rangle$). Formally, the prior predictive density is written as follows:

$$p(D_{pred}) = p(y_{pred_1}, \dots, y_{pred_n}) = \int_{\Theta} p(y_{pred_1}|\Theta) \cdot p(y_{pred_2}|\Theta) \cdots p(y_{pred_N}|\Theta) p(\Theta)\, d\Theta \tag{3.5}$$

In essence, we integrate out the vector of parameters, and we end up with the probability distribution of possible datasets given the priors and the likelihood we have defined, *before we encounter any observations*.

We can completely avoid doing the integration by generating samples from the prior distribution instead. Notice here that each sample is an imaginary or potential dataset.

Here is one way to generate prior predictive distributions:

Repeat the following many times: 1. Take one sample from each of the priors. 2. Plug those samples in the likelihood and generate a dataset $y_{pred_1}, \cdots, y_{pred_n}$.

We can create a function that does this:

```
normal_predictive_distribution <- function(mu_samples, sigma_samples, N_obs) {
  # empty data frame with headers:
  df_pred <- tibble(trialn = numeric(0),
                    rt_pred = numeric(0),
                    iter = numeric(0))
  # i iterates from 1 to the length of mu_samples,
  # which we assume is identical to
  # the length of the sigma_samples:
  for (i in seq_along(mu_samples)) {
    mu <- mu_samples[i]
    sigma <- sigma_samples[i]
    df_pred <- bind_rows(
      df_pred,
      tibble(
        trialn = seq_len(N_obs), #1, 2,... N_obs
        rt_pred = rnorm(N_obs, mu, sigma),
        iter = i
      )
    )
```

```
  }
  df_pred
}
```

The following code produces 1000 samples of the prior predictive distribution of the model that we defined in 3.1.1.1. Although this approach works, it's quite slow (it takes about 5 seconds):

```
tic()
N_samples <- 1000
N_obs <- nrow(df_noreading_data)
mu_samples <- runif(N_samples, 0, 60000)
sigma_samples <- runif(N_samples, 0, 2000)

normal_predictive_distribution(
  mu_samples = mu_samples,
  sigma_samples = sigma_samples,
  N_obs = N_obs
)
```

```
## # A tibble: 361,000 x 3
##   trialn rt_pred  iter
##    <dbl>   <dbl> <dbl>
## 1      1  37891.     1
## 2      2  38266.     1
## 3      3  38339.     1
## 4      4  38988.     1
## 5      5  39062.     1
## # ... with 3.61e+05 more rows
```

```
toc()
```

```
## 4.707 sec elapsed
```

We can create a more efficient function in the following way. With this function, we see an approximately 10-fold increase in speed. Notice that

while the distributions should be the same with both functions, the numbers that we see in the tables won't be, due to the randomness in the process of sampling.

```
normal_predictive_distribution_fast <- function(mu_samples,
                                                sigma_samples,
                                                N_obs) {
  # map_dfr works similarly to lapply, it essentially runs
  # a for-loop, and builds a dataframe with the output.
  # We iterate over the values of mu_samples and sigma_samples
  # simultaneously, and in each iteration we bind a new
  # data frame with N_obs observations.
  map2_dfr(mu_samples, sigma_samples, function(mu, sigma) {
    tibble(
      trialn = seq_len(N_obs),
      rt_pred = rnorm(N_obs, mu, sigma)
    )
  }, .id = "iter") %>%
    # .id is always a string and needs to be converted to a number
    mutate(iter = as.numeric(iter))
}

tic()
(prior_pred <- normal_predictive_distribution_fast(
  mu_samples = mu_samples,
  sigma_samples = sigma_samples,
  N_obs))
```

```
## # A tibble: 361,000 x 3
##    iter trialn rt_pred
##   <dbl>  <int>   <dbl>
## 1     1      1  36807.
## 2     1      2  37132.
## 3     1      3  38655.
## 4     1      4  39613.
## 5     1      5  39348.
## # ... with 3.61e+05 more rows
```

```
toc()
```

```
## 0.358 sec elapsed
```

Figure 3.4 shows the first 18 samples of the prior predictive distribution. These are 18 predicted datasets.

```
prior_pred %>%
  filter(iter <= 18) %>%
  ggplot(aes(rt_pred)) +
  geom_histogram() +
  facet_wrap(~iter, ncol = 3)
```

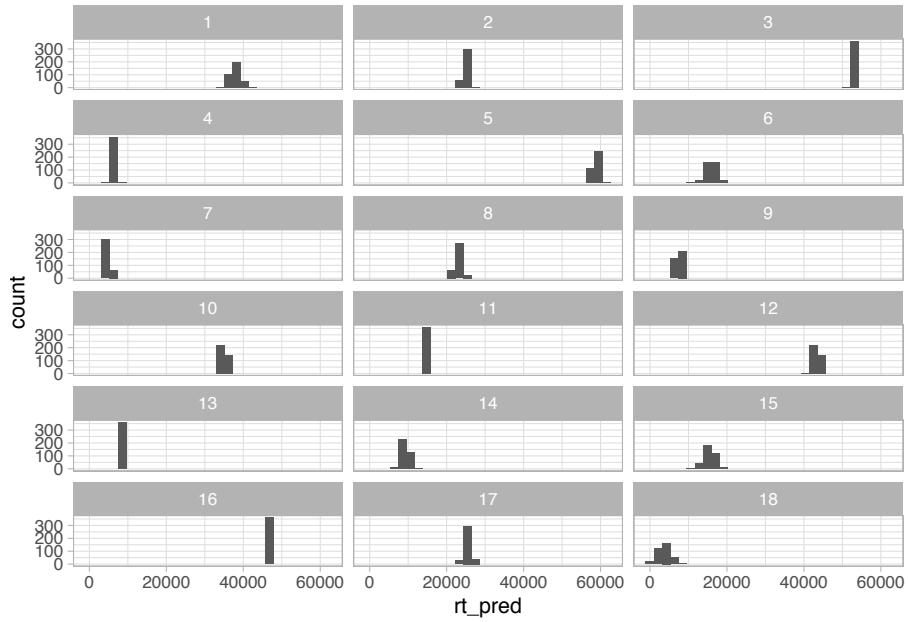


FIGURE 3.4: Eighteen samples from the prior predictive distribution of the model defined in 3.1.1.1.

The prior predictive distribution in Figure 3.4 shows prior datasets that are not realistic: Besides the fact that the datasets show that reaction times distributions are symmetrical–and we know that they are generally

right-skewed–, some datasets present reaction times that are unrealistically long, and worst yet, dataset 18 presents negative press time values.

We can also look at the distribution of statistics here. Even if we don't know beforehand what the data should look like, it's very likely that we have some expectations for possible mean, minimum, or maximum values.

```
prior_pred %>%
  group_by(iter) %>%
  summarize(
    min_rt = min(rt_pred),
    max_rt = max(rt_pred),
    average_rt = mean(rt_pred)
  ) %>%
  # we convert the previous data frame to a long one,
  # where min_rt, max_rt, average_rt are possible values
  # of the columns "stat"
  pivot_longer(cols = ends_with("rt"),
               names_to = "stat",
               values_to = "rt") %>%
  ggplot(aes(rt)) +
  geom_histogram(binwidth = 500) +
  facet_wrap(~stat, ncol = 1)
```

Figure 3.5 shows us that we used much less prior information than what we really had: Our priors were encoding the information that any mean between 0 and 60000 is expected, even though we know that a value close to 0 or to 60000 would be extremely surprising. It should be clear that this results is because we are seeing the effects of our uniform prior on μ. Similarly, maximum values are quite "uniform", spanning a much wider range than what we would expect. Finally, in the distribution of minimum values, we see that negative observations are predicted. This might seem surprising (our prior for μ excluded negative values), but the reason we observe negative values is that the prior is interpreted together with the likelihood (Gelman et al., 2017), and our likelihood is a normal distri-

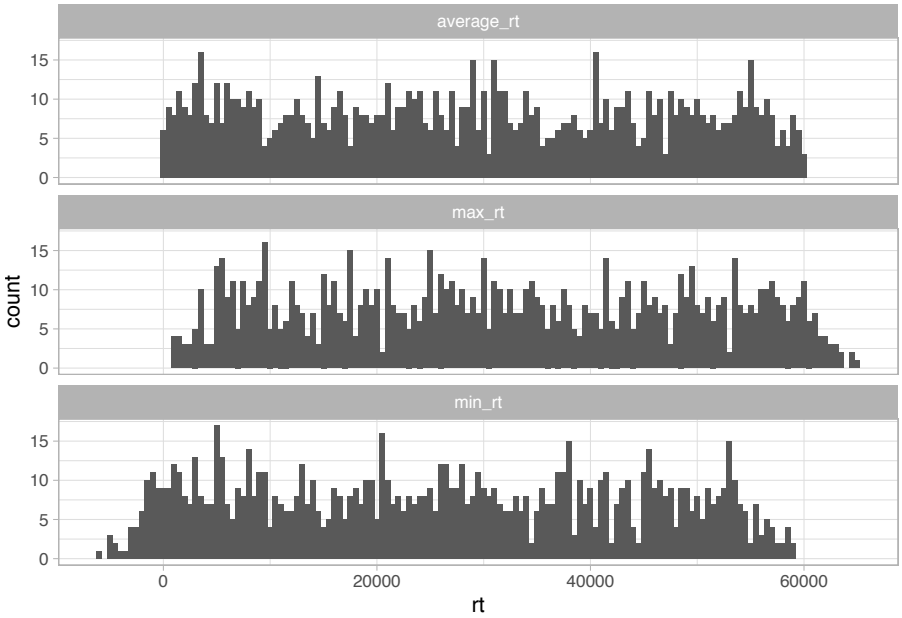


FIGURE 3.5: Prior predictive distribution of averages, maximum, and minimum value of the model defined in 3.1.1.1.

bution, which will allow for negative samples no matter the value of the parameter μ.

To summarize the above discussion, our priors are clearly not very realistic given what we know about reaction times for such a button pressing task. This raises the question: what priors should we have chosen? In the next section, we consider this question.

3.3 The influence of priors: sensitivity analysis

For most cases that we will encounter in this book, there are four main classes of priors that we can choose from:

3.3.1 Flat uninformative priors

One option is to choose priors that are as uninformative as possible. The idea behind this approach is to let the data "speak for itself" and to not bias the statistical inference with "subjective" priors. There are several issues with this approach: First, the prior is as subjective as the likelihood, and in fact, different choices of likelihood might have a much stronger impact on the posterior than different choices of priors. Second, uninformative priors are in general unrealistic because they give equal weight to any value, ignoring the fact that we do have some minimal information about our parameters of interest, at the very least, the order of magnitude (reaction times will be in milliseconds and not days, EEG signals some microvolts and not volts, etc). Finally, uninformative priors make the sampling slower and might lead to problems of convergence. Unless we have a large amount of data, it would be wise to avoid them. In the real data analyses we present in this book,

3.3.2 Regularizing priors

If we don't have much prior information, and we have enough data (what enough means here will presently become clear when we look at specific examples), it is fine to use so-called *regularizing priors*. These are priors that downweight extreme values (that is, they provide regularization), they are not very informative, and mostly let the likelihood dominate in determining the posteriors. These priors are theory-neutral; that is, they do not bias the parameters to values supported by any theory. The idea behind this type of prior is to help to stabilize computation. For many applications, they perform well, but as we will see later, they tend to be problematic if we want to use Bayes factors.

3.3.3 Principled priors

The idea here is to have priors that encode all (or most of) the theory-neutral information that we do have. Since we generally know how our data do and do not look like, we can build priors that truly reflect the properties of potential datasets.

3.3.4 Informative priors

There are cases where we have a lot of prior knowledge, and not much data. In general, unless we have *very* good reasons for having informative priors, we don't want our priors to have too much influence on our posterior. An example where informative priors would be important is when we are investigating a language-impaired population from which we can't get many participants.

These four options constitute a continuum. The model from section 3.1.1.1 falls between flat uninformative and regularizing priors. The priors we used were flat but they allowed for values with at least the right order of magnitude. In practical data analysis situations, we are mostly going to choose priors that fall between regularizing and principled.

to-do: I guess this section could be completed. We should do a bit more justice to people advocating for uninformative priors, and also refer to this idea of uninformative priors that are invariant to transformations. SV: I think more discussion would help beginners. We need to test this section out with readers who are complete beginners.

3.4 Revisiting the button-pressing example with different priors

to-do: SV: could we show the posteriors from the last and this model side by side using ridge plots?

What would happen if we use even wider priors for the model defined in 3.1.1.1? We could assume that every mean between -10^{10} and 10^{10} ms is equally likely. Regarding the standard deviation, we could assume that any value between 0 and 10^{10} is equally likely. We keep the likelihood as it is, and we encode the following priors:

$$\begin{aligned} \mu &\sim Uniform(-10^{10}, 10^{10}) \\ \sigma &\sim Uniform(0, 10^{10}) \end{aligned} \tag{3.6}$$

```
# We fit the model with the default setting of the sampler:
# 4 chains, 2000 iterations with half of them as warmup.
fit_press_unif <- brm(rt ~ 1,
  data = df_noreading_data,
  family = gaussian(),
  prior = c(
    prior(uniform(-10^10, 10^10), class = Intercept),
    prior(uniform(0, 10^10), class = sigma)
  )
)
```

Notice that even with these extremely unrealistic priors the output of the model is virtually identical!

```
fit_press_unif
```

```
##  Family: gaussian
##   Links: mu = identity; sigma = identity
## Formula: rt ~ 1
##    Data: df_noreading_data (Number of observations: 361)
## Samples: 4 chains, each with iter = 2000; warmup = 1000; thin = 1;
##          total post-warmup samples = 4000
##
## Population-Level Effects:
##           Estimate Est.Error l-95% CI u-95% CI Rhat
## Intercept   168.61      1.30   166.09   171.16 1.00
##           Bulk_ESS Tail_ESS
## Intercept     3098     2408
##
## Family Specific Parameters:
##       Estimate Est.Error l-95% CI u-95% CI Rhat
## sigma    25.02      0.97    23.24    26.99 1.00
##       Bulk_ESS Tail_ESS
## sigma     3336     2524
##
## Samples were drawn using sampling(NUTS). For each parameter, Eff.Sample
```

```
## is a crude measure of effective sample size, and Rhat is the potential
## scale reduction factor on split chains (at convergence, Rhat = 1).
```

What would happen if we had used very informative priors? We will assume that means very close to 400 ms are the most likely, and that the standard deviation of the reaction times is very close to 100. We already know that this information is clearly wrong, because we have already seen the output of some models. Notice that the $Normal_+$ notation indicates a normal distribution truncated in zero that only allows positive values:

$$\begin{aligned} \mu &\sim Normal(200, 200) \\ \sigma &\sim Normal_+(0, 500) \end{aligned} \tag{3.7}$$

```
fit_press_inf <- brm(rt ~ 1,
  data = df_noreading_data,
  family = gaussian(),
  prior = c(
    prior(normal(400, 10), class = Intercept),
    # brms knows that SD needs to be bounded by zero:
    prior(normal(100, 10), class = sigma)
  )
)
```

Even in this case, the likelihood mostly dominates and the new estimates are just a couple of milliseconds away from our previous estimates:

```
fit_press_unif
```

```
##  Family: gaussian
##   Links: mu = identity; sigma = identity
## Formula: rt ~ 1
##    Data: df_noreading_data (Number of observations: 361)
## Samples: 4 chains, each with iter = 2000; warmup = 1000; thin = 1;
##          total post-warmup samples = 4000
##
## Population-Level Effects:
##           Estimate Est.Error l-95% CI u-95% CI Rhat
```

```
## Intercept   168.61      1.30   166.09   171.16 1.00
##           Bulk_ESS Tail_ESS
## Intercept     3098     2408
##
## Family Specific Parameters:
##       Estimate Est.Error l-95% CI u-95% CI Rhat
## sigma    25.02      0.97    23.24    26.99 1.00
##       Bulk_ESS Tail_ESS
## sigma     3336     2524
##
## Samples were drawn using sampling(NUTS). For each parameter, Eff.Sample
## is a crude measure of effective sample size, and Rhat is the potential
## scale reduction factor on split chains (at convergence, Rhat = 1).
```

This doesn't mean that priors never matter. When there is enough data, the likelihood will dominate in determining the posterior distributions. However, even in the cases where the likelihood dominates, more accurate priors (i.e., more consistent with our real previous belief about the data) will in general speed-up model convergence.

If we are not sure about the extent to which the posterior is influenced by our priors, we can do a *sensitivity analysis*: we try different priors and either verify that the posterior doesn't change drastically, or report how the posterior is affected by some specific priors (for a published example in psycholinguistics, see Vasishth et al., 2013). We will see later in this book that *sensitivity analysis* becomes crucial for reporting Bayes factors (in section ???); even in cases where the choice of priors does not affect the posterior distribution, it generally affects the Bayes factor.

3.5 Posterior predictive distribution

The prior predictive distribution is a collection of datasets generated from the model (the likelihood and the priors). After we have seen the data and obtained the posterior distributions of the parameters, we can now use the *posterior distributions* to generate future data from the model. In other words, given the posterior distributions of the parameters of the

model, the posterior predictive distribution shows how future data might look like.

Once we have the posterior distribution $p(\Theta \mid y)$, we can derive the predictions based on this distribution:

$$p(D_{pred} \mid y) = \int_{\Theta} p(D_{pred}, \Theta \mid y)\, d\Theta = \int_{\Theta} p(D_{pred} \mid \Theta, y) p(\Theta \mid y)\, d\Theta \tag{3.8}$$

Assuming that past and future observations are conditionally independent given Θ, i.e., $p(D_{pred} \mid \Theta, y) = p(D_{pred} \mid \Theta)$, we can write:

$$p(D_{pred} \mid y) = \int_{\Theta} p(D_{pred} \mid \Theta) p(\Theta \mid y)\, d\Theta \tag{3.9}$$

Note that we are conditioning D_{pred} only on y, we do not condition on what we don't know (Θ); we integrate out the unknown parameters. This posterior predictive distribution is different from the frequentist approach, which gives only a predictive distribution of D_{pred} given our maximum likelihood estimate of θ (a point value). As with the prior predictive distribution, we can avoid the integration by generating samples from the posterior predictive distribution. We can use the same function that we created before, `normal_predictive_distribution_fast`, with the only difference in that instead of sampling `mu` and `sigma` from the priors, we use samples from the posterior.

```
N_obs <- nrow(df_noreading_data)
mu_samples <- posterior_samples(fit_press)$b_Intercept
sigma_samples <- posterior_samples(fit_press)$sigma
normal_predictive_distribution_fast(
  mu_samples = mu_samples,
  sigma_samples = sigma_samples,
  N_obs
)
```

```
## # A tibble: 1,444,000 x 3
##    iter trialn rt_pred
```

```
##   <dbl>  <int>   <dbl>
## 1     1      1    154.
## 2     1      2    170.
## 3     1      3    161.
## 4     1      4    203.
## 5     1      5    181.
## # ... with 1.444e+06 more rows
```

There is a built-in function provided in `brms`, that will give us the posterior predictive distribution: `posterior_predict(fit_press)` provides the predicted reaction times in a matrix, with the number of samples as rows and the number of observations (data-points) as columns.

We can use the posterior predictive distribution to examine the "descriptive adequacy" of our models (Gelman et al., 2014, Chapter 6; Shiffrin et al., 2008); these are called posterior predictive checks, and what we want to establish here is that the posterior predictive data look more or less similar to the observed data. Achieving descriptive adequacy means that the current data could have been generated by the model. While passing a test of descriptive adequacy is not strong evidence in favor of a model, a major failure in descriptive adequacy can be interpreted as strong evidence against a model (Shiffrin et al., 2008). Thus, posterior predictive checking is an important sanity check to assess whether the model behavior is reasonable.

In many cases, we can simply use the plot functions from `brms` and `bayesplot` that take the model as an argument for the visualization posterior predictive checks. For example, we can use `pp_check` to investigate how well the observed distribution of reaction times fit our model based on some number (11 and 100) samples of the posterior predictive distributions; see figures 3.6 and 3.7.

```
pp_check(fit_press, nsamples = 11, type = "hist")
```

```
pp_check(fit_press, nsamples = 100)
```

Notice that the real data is slightly skewed and has no values shorter than 100 ms, while the predictive distributions are centered and symmetrical;

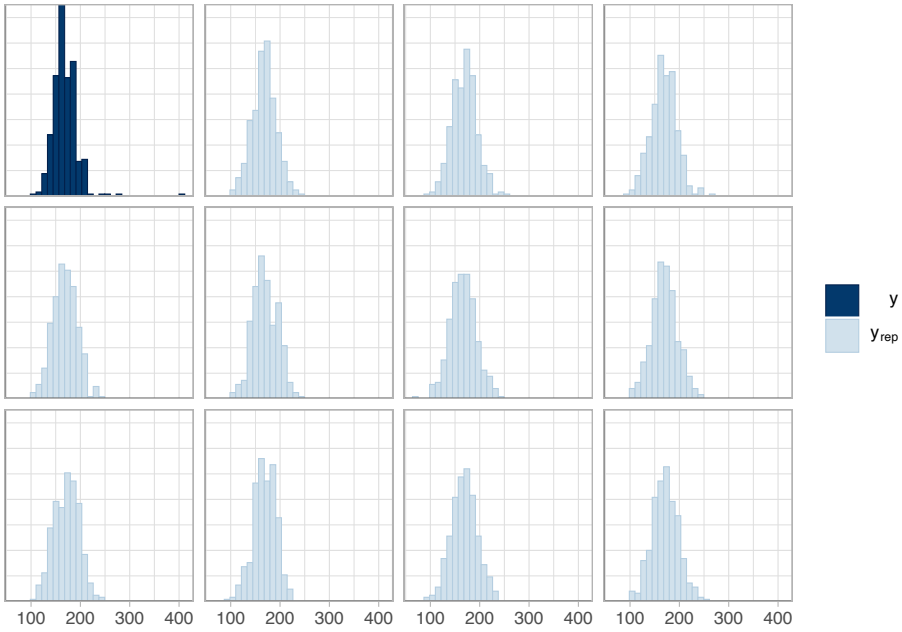


FIGURE 3.6: Eleven samples from the posterior predictive distribution of the model `fit_press`.

see figures 3.6 and 3.7. This posterior predictive check shows a slight mismatch between the observed and predicted data. Can we build a better model? We'll come back to this issue in the next section.

3.5.1 Comparing different likelihoods

Since we know that the reaction times shouldn't be normally distributed, we can choose a more realistic distribution for the likelihood. A good candidate is the log-normal distribution since a variable (such as time) that is log-normally distributed takes only positive real values and is right skewed.

3.5.2 The log-normal likelihood

If y is log-normally distributed, this means that $\log(y)$ is normally distributed.[2] Something important to notice is that the log-normal distribu-

[2] In fact, $\log_e(y)$ or $\ln(y)$, but we'll write it as just $log()$

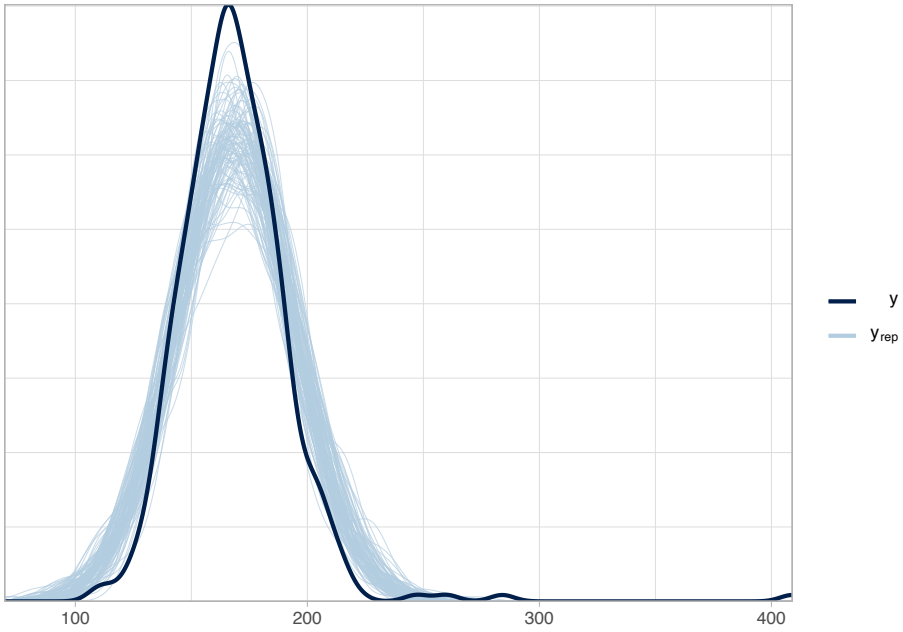


FIGURE 3.7: Posterior predictive check that shows the fit of the model `fit_press` in comparison to datasets from the posterior predictive distribution.

tion is again defined using μ and σ, but these correspond to the mean and standard deviation of the normally distributed logarithm of the data y: $\log(y)$. Thus, when we model some data y using the log-normal likelihood, the parameters μ and σ are on a different scale than the data y.

We can create a log-normal distribution by exponentiating the samples of a normal distribution. See Figure 3.8.

$$\begin{aligned} \log(y) &\sim Normal(\mu, \sigma) \\ y &\sim \exp(Normal(\mu, \sigma)) \\ y &\sim LogNormal(\mu, \sigma) \end{aligned} \tag{3.10}$$

```
mu <- 6
sigma <- 0.5
N <- 500000
```

```r
# Generate N random samples from a log-normal distribution
sl <- rlnorm(N, mu, sigma)
ggplot(tibble(samples = sl), aes(sl)) +
  geom_histogram(binwidth = 50) +
  ggtitle("Log-normal distribution\n") +
  coord_cartesian(ylim = c(0, 70000), xlim = c(0, 2000))
# Generate N random samples from a normal distribution,
# and then exponentiate them
sn <- exp(rnorm(N, mu, sigma))
ggplot(tibble(samples = sn), aes(sn)) +
  geom_histogram(binwidth = 50) +
  ggtitle("Exponentiated samples of\na normal distribution") +
    coord_cartesian(ylim = c(0, 70000), xlim = c(0, 2000))
```

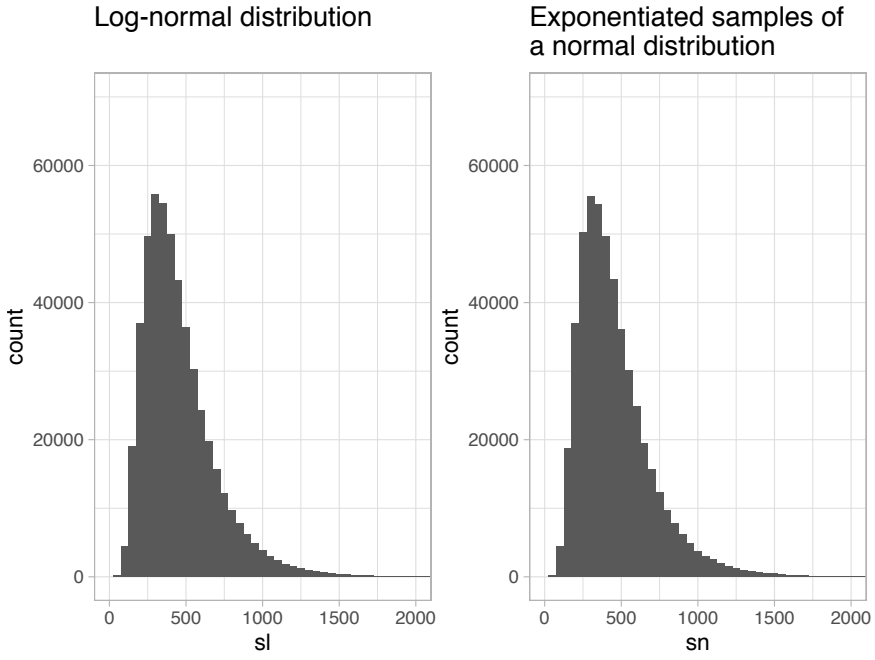


FIGURE 3.8: Two log-normal distributions with the same parameters generated by either generating samples from a log-normal distribution or exponentiating samples from a normal distribution.

3.5.3 Re-fitting a single participant pressing a button repeatedly with a log-normal likelihood

If we assume that reaction times are log-normally distributed, we'll need to change our likelihood function as follows:

$$rt_n \sim LogNormal(\mu, \sigma) \tag{3.11}$$

But now the scale of our priors needs to change! We'll continue with the uniform priors for ease of exposition, even though, as we mentioned earlier, these are not recommended.

$$\begin{aligned} \mu &\sim Uniform(0, 8) \\ \sigma &\sim Uniform(0, 1) \end{aligned} \tag{3.12}$$

Because the parameters are in a different scale than the dependent variable, their interpretation changes and it is more complex than if we were dealing with a linear model that assumes a normal likelihood (location and scale do not coincide with the mean and standard deviation of the log-normal):

- *The location, μ*: In our previous linear model, μ represented the grand mean (or the grand median, or grand mode, since in a normal distribution the three coincide). But now, the grand mean needs to be calculated in the following way, $\exp(\mu + \sigma^2/2)$. Interestingly, the grand median will just be $\exp(\mu)$. We could assume that the grand median, $\exp(\mu)$, represents the underlying time it takes to press the space bar if there would be no noise, that is, if σ would be 0. This also means that the prior of μ is not in milliseconds, but in log(milliseconds).
- *The scale, σ*: This is the standard deviation of the normal distribution of $\log(y)$. The standard deviation of a log-normal distribution with *location* μ and *shape* σ will be $\exp(\mu+\sigma^2/2) \times \sqrt{(}\exp(\sigma^2)-1)$. It's important to notice that, unlike the normal distribution, the spread of the log-normal distribution depends on both μ and σ.

To understand the meaning of our priors in milliseconds scale, we need to take into account both the priors and the likelihood. We can do this by generating a prior predictive distribution. Notice that we can just exponentiate the samples produced by `nor-`

`mal_predictive_distribution_fast()` (or, alternatively, we could have edited the function and replaced `rnorm` for `rlnorm`).

```
N_samples <- 1000
N_obs <- nrow(df_noreading_data)
mu_samples <- runif(N_samples, 0, 8)
sigma_samples <- runif(N_samples, 0, 1)
prior_pred_ln <- exp(normal_predictive_distribution_fast(
  mu_samples = mu_samples,
  sigma_samples = sigma_samples,
  N_obs
))
```

And then we plot the distribution of some representative statistics:

```
prior_pred_ln %>%
  group_by(iter) %>%
  summarize(
    min_rt = min(rt_pred),
    max_rt = max(rt_pred),
    average_rt = mean(rt_pred),
    median_rt = median(rt_pred)
  ) %>%
  pivot_longer(cols = ends_with("rt"), names_to = "stat", values_to = "rt") %>%
  ggplot(aes(rt)) +
  scale_x_continuous("Reaction times in ms",
    trans = "log", breaks = c(0.001, 1, 100, 1000, 10000, 100000)
  ) +
  geom_histogram() +
  facet_wrap(~stat, ncol = 1)
```

While we cannot not generate negative values anymore, since exp(any number) > 0, and these priors might work, we can choose better regularizing priors for our model, such as the following:

$$\begin{aligned} \mu &\sim Normal(6, 1.5) \\ \sigma &\sim Normal_+(0, 1) \end{aligned} \tag{3.13}$$

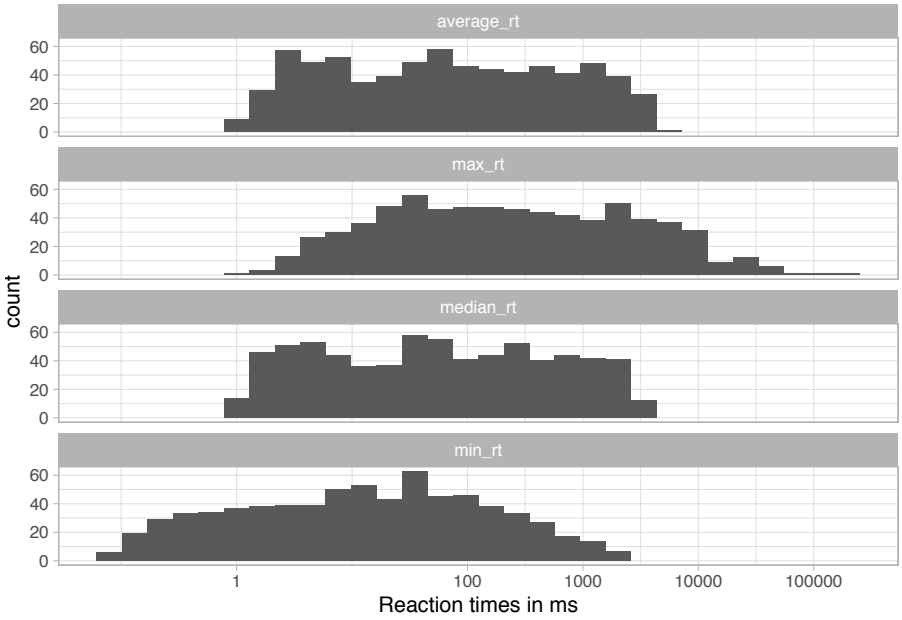


FIGURE 3.9: Prior predictive distribution of averages, maximum, and minimum value of the log-normal model with priors defined in (3.12). Notice that the x-axis is log-transformed.

Notice that while μ can be negative, the dependent variable won't, since the exponent of a negative value, $\exp(\text{some negative value})$, is always greater than 0. Even before generating the prior predictive distributions, we can calculate the values within which we are 95% sure that the expected median of the observations will lie. We do this by looking at what happens at two standard deviations away from the mean of the *prior*, μ, that is $6 - 2 \times 1.5$ and $6 + 2 \times 1.5$, and exponentiating these values:

```
c(
  lower = exp(6 - 2 * 1.5),
  higher = exp(6 + 2 * 1.5)
)
```

```
##  lower higher
##     20   8103
```

This means that our prior for μ is still not too informative (these are me-

dians; the actual values generated by the distribution can be much more spread out). We plot the distribution of some representative statistics in Figure 3.10.

```
N_samples <- 1000
N_obs <- nrow(df_noreading_data)
mu_samples <- rnorm(N_samples, 6, 1.5)
sigma_samples <- rtnorm(N_samples, 0, 1, a = 0)
prior_pred_ln_better <- exp(normal_predictive_distribution_fast(
  mu_samples = mu_samples,
  sigma_samples = sigma_samples,
  N_obs
))
prior_pred_ln_better %>%
  group_by(iter) %>%
  summarize(
    min_rt = min(rt_pred),
    max_rt = max(rt_pred),
    average_rt = mean(rt_pred),
    median_rt = median(rt_pred)
  ) %>%
 pivot_longer(cols = ends_with("rt"),
              names_to = "stat", values_to = "rt") %>%
  ggplot(aes(rt)) +
  scale_x_continuous(trans = "log", breaks = c(0.001, 1, 100, 1000, 10000, 100000)) +
  geom_histogram() +
  facet_wrap(~stat, ncol = 1) +
  coord_cartesian(xlim = c(0.001, 300000))
```

We see that the priors that we are using are still too uninformative. We could do more iterations of choosing priors and generating posterior predictive distributions until we have priors that generate realistic data. However, for most cases, priors that generate data that whose statistics (mean, median, min, max, etc.) lie roughly in the correct order of magnitude are going to be acceptable.

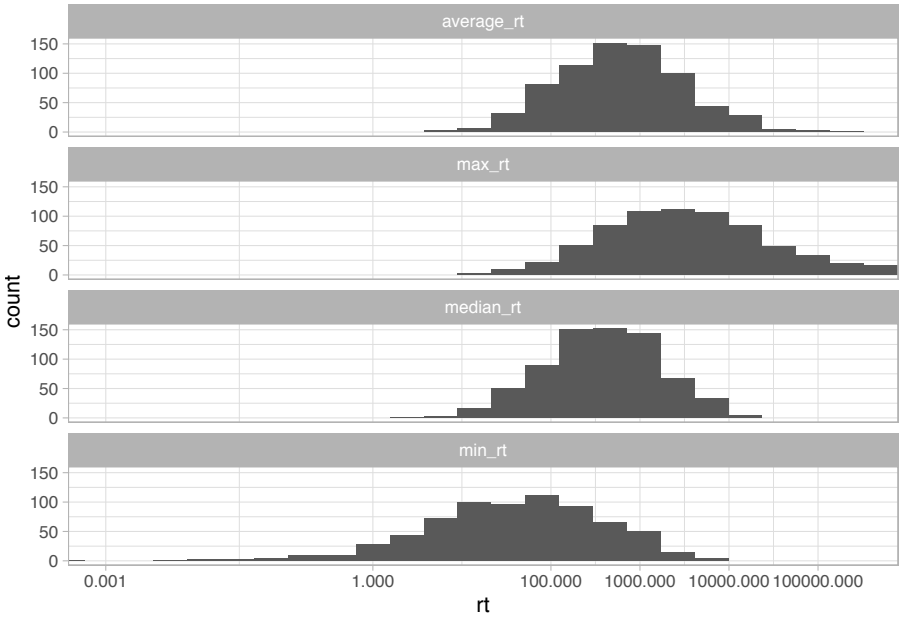


FIGURE 3.10: Prior predictive distribution of averages, maximum, and minimum value of the log-normal model with priors defined in (3.13). Notice that the x-axis is log-transformed.

We can fit the model now, but notice that we need to specify that the family is `lognormal()`. In our first example, we had used the family `gaussian()`.

```
fit_press_ln <- brm(rt ~ 1,
  data = df_noreading_data,
  family = lognormal(),
  prior = c(
    prior(normal(6, 1.5), class = Intercept),
    prior(normal(0, 1), class = sigma)
  )
)
```

When we look at the summary of the posterior, the parameters are in log-scale:

```
fit_press_ln
```

```
##  Family: lognormal
##   Links: mu = identity; sigma = identity
## Formula: rt ~ 1
##    Data: df_noreading_data (Number of observations: 361)
## Samples: 4 chains, each with iter = 2000; warmup = 1000; thin = 1;
##          total post-warmup samples = 4000
##
## Population-Level Effects:
##           Estimate Est.Error l-95% CI u-95% CI Rhat
## Intercept     5.12      0.01     5.10     5.13 1.00
##           Bulk_ESS Tail_ESS
## Intercept     4198     2798
##
## Family Specific Parameters:
##       Estimate Est.Error l-95% CI u-95% CI Rhat
## sigma     0.13      0.00     0.13     0.14 1.00
##       Bulk_ESS Tail_ESS
## sigma     2906     2732
##
## Samples were drawn using sampling(NUTS). For each parameter, Eff.Sample
## is a crude measure of effective sample size, and Rhat is the potential
## scale reduction factor on split chains (at convergence, Rhat = 1).
```

If we want to know how long does it take to press the space bar in milliseconds, we need to transform the μ (or `Intercept` in the model) to milliseconds. Since we know that the median of the log-normal distribution is $exp(\mu)$, we do the following to calculate an estimate in milliseconds:

```
estimate_ms <- exp(posterior_samples(fit_press_ln)$b_Intercept)
```

If we want to know the mean and 95% credible interval, we do the following:

```
c(mean = mean(estimate_ms), quantile(estimate_ms, probs = c(.025, .975)))
```

```
## mean 2.5%  98%
##  167  165  169
```

We can now verify whether our predicted datasets look similar to the real dataset. See Figure 3.11; compare this with the earlier Figure 3.7.

```
pp_check(fit_press_ln, nsamples = 100)
```

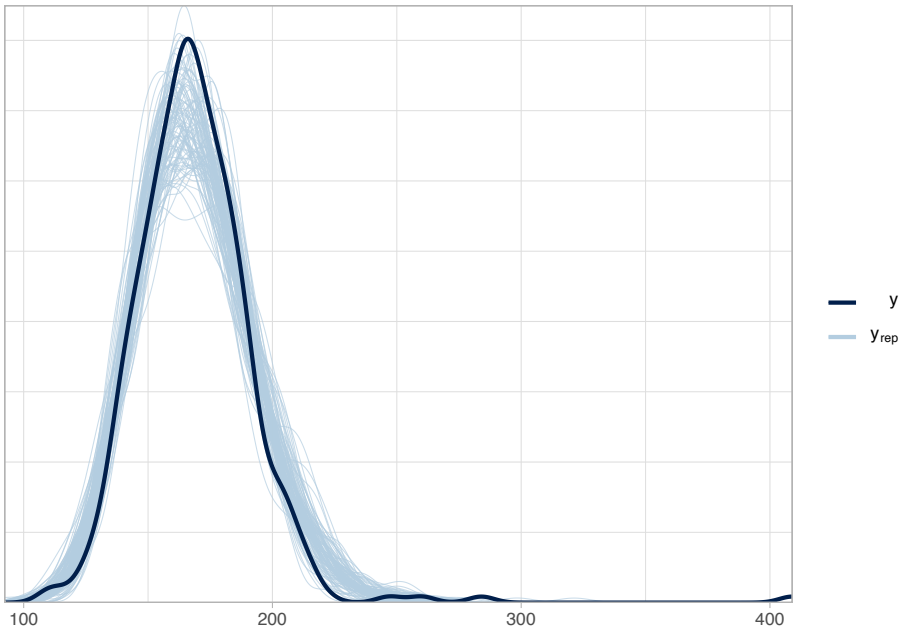


FIGURE 3.11: Posterior predictive distribution of `m_noreading_ln`

Are the posterior predicted data now more similar to the real data, compared to the case where we had a Normal likelihood?

It seems so, but it's not easy to tell. Another way to examine this would be to look at the distribution of summary statistics. We compare the distribution of representative summary statistics for the datasets generated by different models and compare them to the observed statistics. We suspect that the normal distribution would generate reaction times that are

too fast (since it's symmetrical) and that the log-normal distribution may capture the long tail better than the normal model. Based on our hunch, we compute the distribution of minimum and maximum values for the posterior predictive distributions, and we compare them with the minimum and maximum value respectively in the data.

```
pp_check(fit_press, type = "stat", stat = "min") + ggtitle("Normal model")
pp_check(fit_press_ln, type = "stat", stat = "min") + ggtitle("Log-normal model")
```

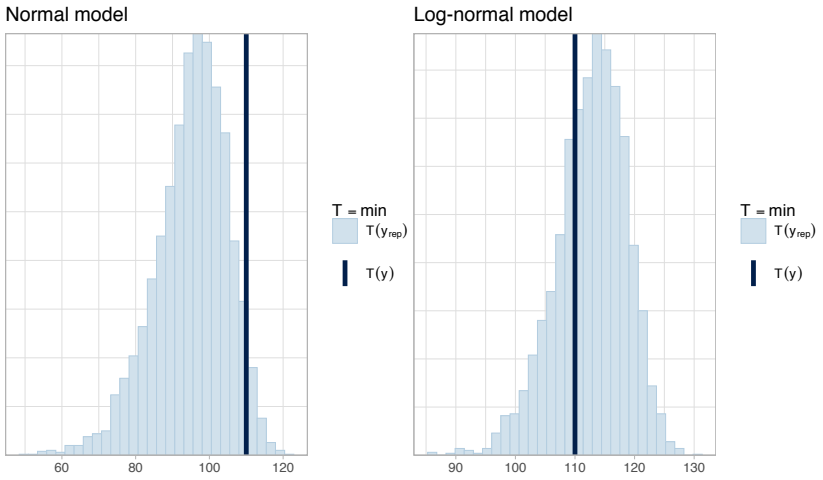


FIGURE 3.12: Distribution of minimum values in a posterior predictive check. The minimum in the data is 110 ms.

```
pp_check(fit_press, type = "stat", stat = "max") + ggtitle("Normal model")
pp_check(fit_press_ln, type = "stat", stat = "max") + ggtitle("Log-normal model")
```

Figure 3.12 shows that the log-normal likelihood does a slightly better job since the minimum value is contained in the bulk of the log-normal distribution and in the tail of normal one. Figure 3.13 shows that both models are unable to capture the maximum value of the observed data. One explanation for this is that the log-normal-ish observations in our data are being generated by the task of pressing as fast as possible, while the observations with long reaction times are being generated by lapses of attention.

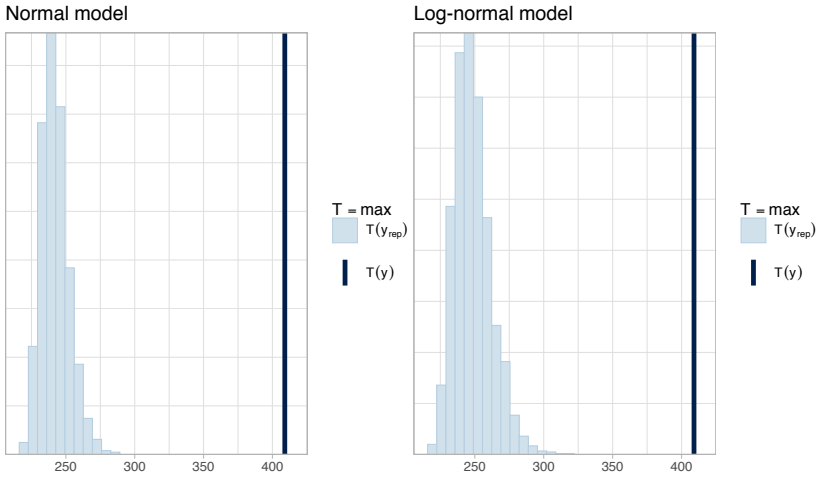


FIGURE 3.13: Distribution of maximum values in a posterior predictive check. The maximum in the data is 409 ms.

3.6 Summary

3.7 Further reading

On the topic of priors:

- Chapter 5 of Lunn D, Jackson C, Spiegelhalter DJ, Best N,, Thomas A (2012). *The BUGS book: A practical, introduction to Bayesian analysis*, volume 98., CRC Press.
- Gelman A, Simpson D, Betancourt M (2017). "The, prior can often only be understood in the, context of the likelihood." *Entropy*, *19*(10),, 555. doi: 10.3390/e19100555 (URL:, https://doi.org/10.3390/e19100555), <URL:, https://www.mdpi.com/1099-4300/19/10/555>.
- Simpson D, Rue H, Riebler A, Martins TG, Sørbye, SH (2017). "Penalising Model Component, Complexity: A Principled, Practical Approach to, Constructing Priors." *Statistical Science*,, *32*(1), 1-28. ISSN 0883-4237, 2168-8745, doi:, 10.1214/16-STS576 (URL:, https://doi.org/10.1214/16-STS576), <URL:, https://projecteuclid.org/euclid.ss/1491465621>.

- Prior Distributions for rstanarm Models in https://mc-stan.org/rstanarm/articles/priors.html
- Prior Choice Recommendations in https://github.com/stan-dev/stan/wiki/Prior-Choice-Recommendations

3.8 Exercises

1. Can you find priors on μ that have an undesirable impact in our estimates for the model with normal likelihood? And for the model with the log-normal likelihood?
2. For the log-normal model, change the prior of σ to .5 and generate prior predictive distributions.
3. For the log-normal model, try to find sensible priors with prior predictive checks.
4. For the log-normal model, what is the mean (rather than median) time that takes to press the space bar, what is the standard deviation of the reaction times in milliseconds?

3.9 Appendix

3.9.1 Generating prior predictive distributions with `brms`

`brms` can generate prior predictive distributions for us ignoring the data, by using `sample_prior = "only"`. Since we are not fitting any model, we can ignore the messages and warnings, and we set only one chain with 1000 iterations and no warmup. `posterior_predict(...)` and `predict(..., summary = FALSE)` will return an array of N samples by N observations of prior predictions (this is because of `sample_prior = "only"`), and we'll need to manipulate the array to get the desired data frame.

```
fit_press_prior <- brm(rt ~ 1,
  family = gaussian(),
  prior = c(
```

```
    prior(uniform(0, 60000), class = Intercept),
    prior(uniform(0, 2000), class = sigma)
  ),
  sample_prior = "only",
  warmup = 0,
  chains = 1,
  iter = 1000,
  data = df_noreading_data
)

# predict with summary = FALSE returns a N_samples * N_obs array:
prior_pred <- predict(fit_press_prior, summary = FALSE) %>%
  # array_banch converts it in a list of N_samples elements,
  # and in each element a dataset of N_obs:
  array_branch(1) %>%
  # map_dfr loops over the N_sample elements of the list and builds a data frame
  map_dfr(~ tibble(rt_pred = .x, trialn = seq_along(.x)), .id = "sample_n") %>%
  # .id is always a string and needs to be converted
  mutate(sample_n = as.numeric(sample_n))
```

4

Bayesian regression models

We generally run experiments because we are interested in the relationship between two or more observables. A regression will tell us how our *dependent variable*, also called the *response* or *outcome variable* (e.g., pupil size, reaction times, accuracy, etc) is affected by one or many *independent variables*, *predictors*, or *explanatory variables*. Predictors can be categorical (e.g., male or female), ordinal (first, second, third, etc), or continuous. We will assume that our predictors are continuous in this chapter, and we will refer to them (mostly) as *covariates*. Unfortunately, many times it will happen that the same concept has different names, and a name can be associated with different concepts (mostly depending on the context). *Covariates* are sometimes used to refer to control variables; we won't use them with this meaning in the book, but it's a good idea to bear this in mind.

to-do: This could be expanded much more.

4.1 A first linear model: Does attentional load affect pupil size?

We'll look at the effect of cognitive processing on human pupil size to illustrate the use of Bayesian linear regression models. Although pupil size is mostly related to the amount of light that reaches the retina or the distance to a perceived object, pupil sizes are also systematically influenced by cognitive processing: It has been found that increased cognitive load leads to an increase in the pupil size (for a review, see Mathot, 2018).

For this example, we'll use the data of one participant's pupil size

of the control experiment of Wahn et al. (2016) averaged by trial.[1] In this experiment, a participant covertly tracked between zero and five objects among several randomly moving objects on a computer screen. In this task, called multiple object tracking [or `MOT;@pylyshynTrackingMultipleIndependent1988`[2]] task, several objects appear in the screen, and a subset of them are indicated as "targets" at the beginning. Then, the objects start moving randomly across the screen and become indistinguishable. After several seconds, the objects stop moving and the participant needs to indicate which objects were the targets. See also Figure 4.1. Our research goal is to examine how the number of moving objects being tracked, that is how the attentional load, affects pupil size.

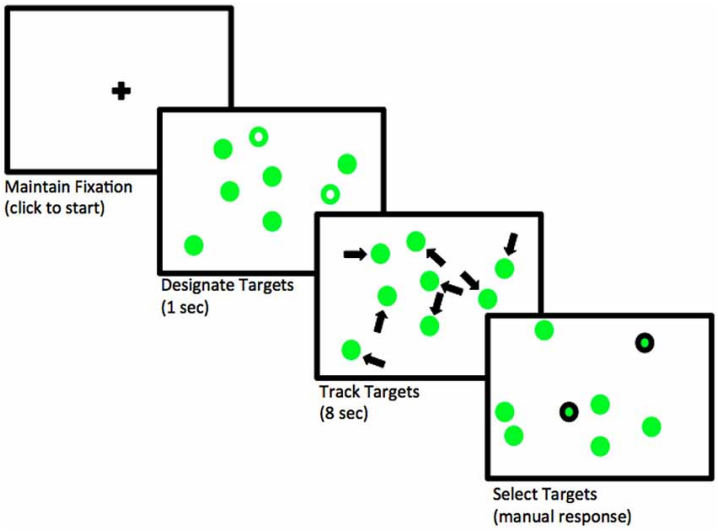


FIGURE 4.1: Flow of events in a trial where two objects needs to be tracked. Adapted from Blumberg et al. (2015); licensed under CC BY 4.0.

4.1.1 Likelihood and priors

We will model pupil size as normally distributed, because we are not expecting a skew, and we have no further information available about the

[1] The full dataset can be found in `https://osf.io/z43dz/`. We show our preprocessing in the appendix of this chapter, section 4.7.1.

[2] `mailto:MOT;@pylyshynTrackingMultipleIndependent1988`

distribution of pupil sizes. (Notice that pupil sizes cannot be of size zero or negative, so we know for sure that this choice is not exactly right.) For simplicity, we are also going to assume a linear relationship between load and the pupil size.

Let's summarize our assumptions:

1. There is some average pupil size represented by α.
2. The increase of attentional load has a linear relationship with pupil size, determined by β.
3. There is some noise in this process, that is, variability around the true pupil size i.e., a scale, σ.
4. The noise is normally distributed.

Our likelihood will be as follows:

$$p\_size_n \sim Normal(\alpha + c\_load_n \cdot \beta, \sigma) \tag{4.1}$$

where n indicates the observation number with $n = 1 \dots N$

This means that the formula that we'll use in `brms` will be `p_size ~ 1 + c_load`, where `1` represents the intercept, α, which doesn't depend on a covariate or predictor, and `c_load` is our covariate that is multiplied by β. We will generally indicate with the prefix `c_`, that a covariate (in this case load) is centered (i.e., we subtract from each value the mean of all values). If load is centered, the intercept represents the pupil size at the average load in the experiment (because at the average load, the centered load is zero, and then $\alpha + 0 \cdot \beta$). Alternatively, if the load would not have been centered (i.e., starts with no load, then one, two, etc), then the intercept would represent the pupil size when there is no load. Although this formula would be enough to fit a frequentist model with `lm(p_size ~ 1 + c_load, dataset)`, when we fit a Bayesian model, we have to specify priors for each of the parameters.

For setting the priors, we need information about pupil sizes. While we might know that pupil sizes range between 2 and 5 millimeters (**?**), this experiment was conducted with the Eyelink-II eyetracker which measures the pupils in arbitrary units (Hayes and Petrov, 2016). If this is our first analysis of pupil size, before setting up the priors, we'll need to look at

some measures of pupil size. (If we had analyzed this type of data before, we could also look at estimates from previous experiments). Fortunately, we have some measurements of the same participant with no attentional load for the first 100ms, each 10 ms, in `pupil_pilot.csv`: This will give us some idea about the order of magnitude of our dependent variable.

```
df_pupil_pilot <- read_csv("./data/pupil_pilot.csv")
df_pupil_pilot$p_size %>% summary()
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##     852     856     862     861     866     868
```

With this information we can set a regularizing prior for α. We center the prior around 1000 to be in the right order of magnitude.[3] Since we don't know how much pupil sizes are going to vary by load yet, we include a rather wide prior by defining it as a normally distribution and setting its standard deviation as 500.

$$\alpha \sim Normal(1000, 500) \tag{4.2}$$

Given that our covariate load is centered, with the prior for α, we are saying that we suspect that the average pupil size for the average load in the experiment will be in a 95% central interval limited by approximately $1000 \pm 2 \cdot 500 = [20, 2000]$ units. We can caclulate this in R using the `qnorm` function:

```
c(qnorm(.025, 1000,500), qnorm(.975, 1000, 500))
```

```
## [1]   20 1980
```

We know that the measurements of the pilot data are strongly correlated because they were taken together just some milliseconds apart. For this reason, they won't tell us how much the pupil size can vary. We set up a quite weak prior for σ that encodes our lack of precise information: σ

[3] The average pupil size will probably be higher than 800, since this measurement was with no load, but, in any case, the exact number won't matter, any mean between 500-1500 would be fine if the standard deviation is large.

is surely larger than zero and has to be in the order of magnitude of the pupil size with no load.

$$\sigma \sim Normal_{+}(0, 1000) \tag{4.3}$$

With this prior for σ, we are saying that we expect that the standard deviation of the pupil sizes should be in the following 95% central interval. (Notice that we use `qtnorm(..., a = 0)` and not `qnorm()`).

```
c(qtnorm(.025, 0, 1000, a = 0), qtnorm(.975, 70,1000, a = 0))
```

```
## [1]   31 2290
```

We still need to set a prior for β, the change in pupil size produced by the attentional load. Given that pupil size changes are not easily perceptible (we don't see them in our day-to-day life), we expect them to be much smaller than the pupil size, so we use the following prior:

$$\beta \sim Normal(0, 100) \tag{4.4}$$

With the prior of β, we are saying that we don't really know if the attentional load will increase or even decrease the pupil size (notice that is centered in zero), but we do know that one unit of load (that is one more object to track) will potentially change the pupil size in a way that is consistent with the following 95% central interval.

```
c(qnorm(.025, 0,100), qnorm(.975, 0,100))
```

```
## [1] -196  196
```

That is, we don't expect changes in size that increase or decrease the pupil size in more than 200 units.

to-do: maybe prior predictive distributions here??

4.1.2 The `brms` model

Before fitting the `brms` model, we load the data and center the predictor `load`:

```
df_pupil_data <- read_csv("data/pupil.csv")
df_pupil_data <- df_pupil_data %>%
    mutate(c_load = load - mean(load))
df_pupil_data
```

```
## # A tibble: 41 x 4
##   trial  load p_size c_load
##   <dbl> <dbl>  <dbl>  <dbl>
## 1     1     2  1021. -0.439
## 2     2     1   951. -1.44
## 3     3     5  1064.  2.56
## 4     4     4   913.  1.56
## 5     5     0   603. -2.44
## # ... with 36 more rows
```

Now we can fit the `brms` model:

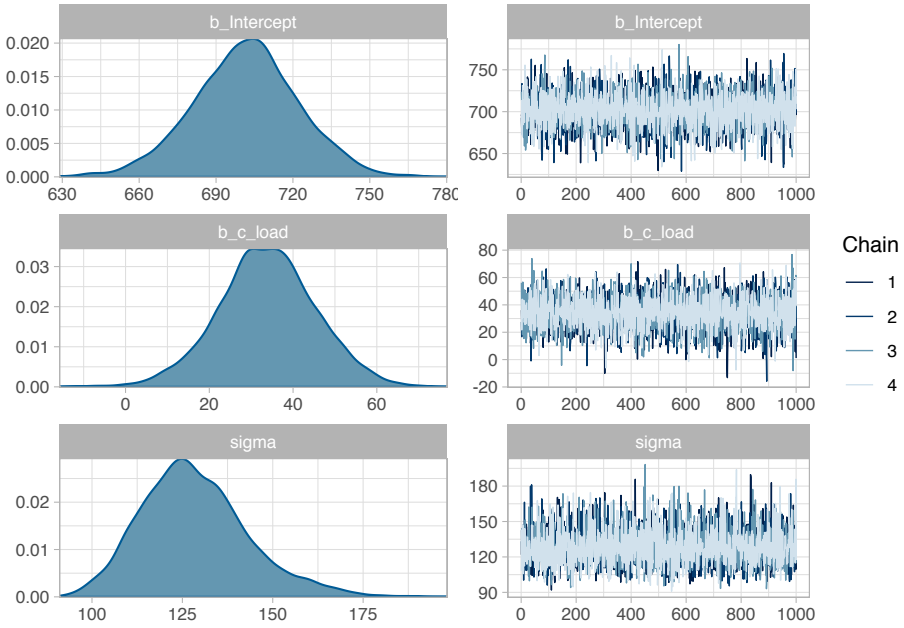
```
fit_pupil <- brm(p_size ~ 1 + c_load,
                 data = df_pupil_data,
                 family = gaussian(),
                 prior = c(
                     prior(normal(1000, 500), class = Intercept),
                     prior(normal(0, 1000), class = sigma),
                     prior(normal(0, 100), class = b, coef = c_load)
                 ))
```

The only difference from our previous models is that we now have a predictor in the formula and in the priors. Priors for predictors are indicated with `class = b`, and the specific predictor with `coef = c_load`. If we want to set the same priors to different predictors we can omit the argument `coef`. We can remove the `1` of the formula, and `brm()` will fit the exact same model as when we specify `1` explicitly. If we really want to remove the in-

tercept we indicate this with `0 +...` or `-1 +...`. See also the box 4.1 for more details about the treatment of the intercepts by `brms`.

We can inspect the output of our model now:

```
plot(fit_pupil)
```




```
fit_pupil
```

```
##  Family: gaussian
##   Links: mu = identity; sigma = identity
## Formula: p_size ~ 1 + c_load
##    Data: df_pupil_data (Number of observations: 41)
## Samples: 4 chains, each with iter = 2000; warmup = 1000; thin = 1;
##          total post-warmup samples = 4000
##
## Population-Level Effects:
##           Estimate Est.Error l-95% CI u-95% CI Rhat
## Intercept   701.68     20.56   660.21   741.57 1.00
## c_load       33.82     11.70    10.26    56.83 1.00
```

```
##           Bulk_ESS Tail_ESS
## Intercept     3444     2684
## c_load        3769     2912
##
## Family Specific Parameters:
##       Estimate Est.Error l-95% CI u-95% CI Rhat
## sigma   128.49     14.82   103.07   161.81 1.00
##       Bulk_ESS Tail_ESS
## sigma     2946     2822
##
## Samples were drawn using sampling(NUTS). For each parameter, Eff.Sample
## is a crude measure of effective sample size, and Rhat is the potential
## scale reduction factor on split chains (at convergence, Rhat = 1).
```

We discuss how we could communicate the relevant information in the next section.

Box 4.1. ***Intercepts in `brms`***

When we set up a prior for the intercept in `brms`, we actually set a prior for an intercept given that all the predictors are centered. This did not matter in our previous examples because we centered our predictor (or we had none), but it might matter if we want to have uncentered predictors. In the design we are discussing, a non-centered predictor of load will mean that the intercept, α, has a straightforward interpretation (in many cases, however, an intercept with a non-centered predictor won't have a straightforward interpretation): the pupil size when there is no attention load.

We might be more sure about prior values for the no load condition, and we want to set the following prior to our new α: $Normal(800, 200)$. In this case, we should fit the following model:

```
fit_pupil_non_centered <- brm(p_size ~ 0 + intercept + load,
                  data = df_pupil_data,
                  family = gaussian(),
                  prior = c(
                      prior(normal(800, 200), class = b, coef = intercept),
                      prior(normal(0, 1000), class = sigma),
                      prior(normal(0, 100), class = b, coef = load)
                  ))
```

Notice that we remove the regular centered intercept by adding `0` to the formula, and we replace it with the "actual" intercept we want to set priors to with `intercept`—this is a reserved word, and thus we cannot name any predictor with this name. This new parameter is also of the class `b`, so its prior needs to be defined accordingly.

The output below shows that, as expected, while the posterior for the intercept has changed noticeably, the posterior for the effect of load remains virtually unchanged.

```
posterior_summary(fit_pupil_non_centered)
```

```
##             Estimate Est.Error   Q2.5 Q97.5
## b_intercept      623      34.3  557.3   691
## b_load            33      11.7    9.8    56
## sigma            129      15.2  104.0   163
## lp__            -271       1.2 -274.1  -270
```

Notice the following potential pitfall. A model like the one below will fit a non-centered load predictor, but will assign a prior of $Normal(800, 200)$ to the intercept of a *centered* model, $\alpha_{centered}$, and not the current intercept, α.

```
fit_pupil_wrong <- brm(p_size ~ 1 + load,
                      data = df_pupil_data,
                      family = gaussian(),
                      prior = c(
                          prior(normal(800, 100), class = Intercept),
                          prior(normal(0, 1000), class = sigma),
                          prior(normal(0, 100), class = b, coef = load)
                      ))
```

What does it mean to set a prior to $\alpha_{centered}$ in a model that *doesn't* include $\alpha_{centered}$?

Notice that the fitted values of the non-centered model and the centered one are identical, that is, the expected values of the response distribution without the residual error (when $\sigma = 0$) are identical for both models:

$$\alpha + load_n \cdot \beta = \alpha_{centered} + (load_n - mean(load)) \cdot \beta \quad (4.5)$$

The left side of Equation (4.5) refers to the fitted values based on our current non-centered model, and the right side refers to the fitted values based on the centered model. We can re-arrange terms to understand what is the effect of a prior on $\alpha_{centered}$ in our model that *doesn't* include $\alpha_{centered}$.

$$\begin{aligned} \alpha + load_n \cdot \beta &= \alpha_{centered} + load_n \cdot \beta - mean(load) \cdot \beta \\ \alpha &= \alpha_{centered} - mean(load) \cdot \beta \\ \alpha + mean(load) \cdot \beta &= \alpha_{centered} \end{aligned} \quad (4.6)$$

That means that we are actually setting our prior to $\alpha + mean(load) \cdot \beta$. When β is very small, and the prior for α is very wide, we might hardly notice the difference between setting a prior to $\alpha_{centered}$ or to our actual α in a non-centered model (especially if the likelihood dominates anyway). But it's a

good idea to pay attention to what are the parameters we are setting priors to.

4.1.3 How to communicate the results?

We want to answer our research question "What is the effect of attentional load on the participant's pupil size?" For that we'll need to examine what happens with β, which is `c_load` in the summary of `brms`. The summary of the posterior tells us that the most likely values of β will be around the mean of the posterior, 33.82, and we can be 95% certain that the true value of β *given the model and the data* lies between 10.26 and 56.83.

We see that as the attentional load increases, the pupil size of the participant becomes larger. If we want to determine how likely it is that the pupil size increased rather than decreased, we can examine the proportion of samples above zero. (Notice that the intercept and the slopes, are always preceded by `b_` in `brms`. One can see all the names of parameters being estimated with `parnames()`.)

```
mean(posterior_samples(fit_pupil)$b_c_load > 0)
```

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

Take into account that this probability ignores the possibility of the participant not being affected at all by the manipulation, this is because $P(\beta = 0) = 0$, we'll come back to this issue in the model comparison section @ref(sec:??).

4.1.4 Descriptive adequacy

Our model converged and we obtained a posterior distribution, there is, however, no guarantee that our model was adequate to represent our data. We can use posterior predictive checks to verify this.

Sometimes it's useful to build our own posterior predictive check to visualize the fit of our model, as opposed to use the `pp_check` functions as we did before in section 3.5. For example, here we use `posterior_predict()` to generate 1000 posterior predictive distributions, and we convert them from an array to a long data frame.

```
# we start from an array of 1000 samples by 41 observations
df_pupil_pred <- posterior_predict(fit_pupil, nsamples = 1000) %>%
    # we convert it to a list of length 1000, with 41 observations in each element:
    array_branch(margin = 1) %>%
    # We iterate over the elements (the predicted distributions)
    # and we convert them into a long data frame similar to the data,
    # but with an extra column `iter` indicating from which iteration
    # the sample is coming from.
    map_dfr( function(yrep_iter) {
        df_pupil_data %>%
            mutate(p_size = yrep_iter)
    }, .id = "iter") %>%
    mutate(iter = as.numeric(iter))
```

Then we plot 100 of the densities of the predicted distributions in blue, and the distribution of our data in black for the five levels of load in Figure 4.2. We don't have enough data to derive a strong conclusion: Notice that both the predictive distributions and our data look very wide, and it hard to tell if the distribution of the observations could have been generated by our model. For now we can say that it doesn't look too bad.

```
df_pupil_pred %>% filter(iter < 100) %>%
    ggplot(aes(p_size, group=iter)) +
    geom_line(alpha = .05, stat="density", color = "blue") +
    geom_density(data=df_pupil_data, aes(p_size),
                 inherit.aes = FALSE, size =1)+
    geom_point(data=df_pupil_data, aes(x=p_size, y = -0.001), alpha =.5,
               inherit.aes = FALSE)+
    coord_cartesian(ylim=c(-0.002, .01))+
    facet_grid(load ~ .)
```

We can instead look at the distribution of a statistic, such as mean pupil size by load:

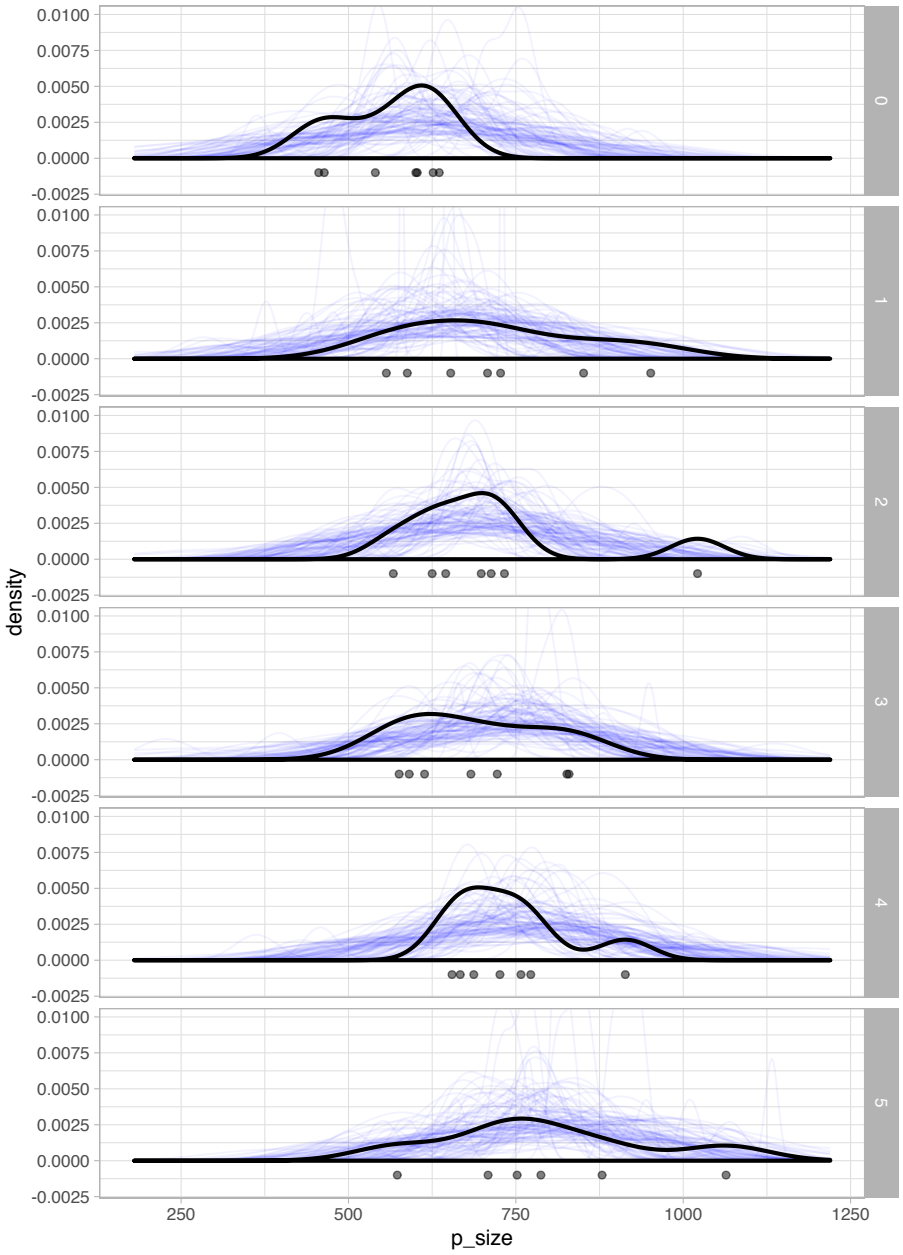


FIGURE 4.2: The plot shows 100 predicted distributions in blue density plots, the distribution of pupil size data in black density plots, and the observed pupil sizes in black dots for the five levels of attentional load.

```
# predicted means:
df_pupil_pred_summary <- df_pupil_pred %>%
    group_by(iter, load) %>%
    summarize(av_p_size = mean(p_size))
# observed means:
df_pupil_summary <- df_pupil_data %>%
    group_by(load) %>%
    summarize(av_p_size = mean(p_size))
```

```
ggplot(df_pupil_pred_summary, aes(av_p_size)) +
    geom_histogram(alpha=.5)+
    geom_vline(aes(xintercept= av_p_size),data= df_pupil_summary)+
    facet_grid(load ~ .)
```

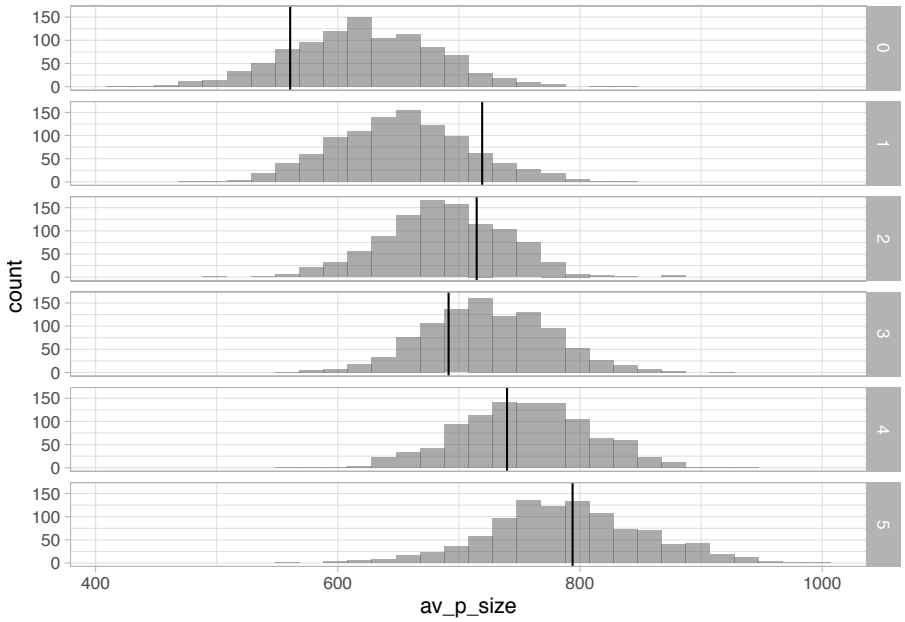


FIGURE 4.3: Distribution of posterior predicted means in gray and observed pupil size means in black lines by load.

Figure 4.3 shows that the observed means for no load and for a load of two are falling in the tails of the distributions. While our model predicts

a monotonic increase of pupil size, the data might be indicating that the relevant difference is between (i) no load, (ii) a load between two and three, and then (iii) a load of four, and (iv) of five. However, given the uncertainty in the posterior predictive distributions and that the observed means are contained somewhere in the predicted distributions, it could be the case that we are overinterpreting noise.

4.2 Log-normal model: Does trial affect pupil size?

Let us revisit the small experiment from section 3.1.1.1, where a participant repeatedly pressed the space bar as fast as possible, without paying attention to the stimuli. We want to know whether the participant tended to speedup (practice effect) or slowdown (fatigue effect) while pressing the space bar.

4.2.1 Likelihood and priors for the log-normal model

If we assume that reaction times are log-normally distributed, we could fit a likelihood such as the following:

$$rt_n \sim LogNormal(\alpha + c\_trial_n \cdot \beta, \sigma) \tag{4.7}$$

where $n = 1, \dots, N$, and rt is the dependent variable (reaction times in milliseconds). The variable N represents the total number of data points.

We use the same priors as in section 3.5.3 for α (which is equivalent to μ in the previous model) and for σ.

$$\begin{aligned} \alpha &\sim Normal(6, 1.5) \\ \sigma &\sim Normal_+(0, 1) \end{aligned} \tag{4.8}$$

We still need a prior for β, but notice that effects are multiplicative rather than additive when we assume a log-normal likelihood and that means that we need to take into account α in order to interpret β; see Box 4.2. We are going to try to understand how all our priors interact together generating some prior predictive distributions. We start with the follow-

ing prior centered in zero, a prior agnostic regarding the direction of the effect, which allows for both a slowdowns ($\beta > 0$) or a speedups ($\beta < 0$):

$$\beta \sim Normal(0, 1) \tag{4.9}$$

We can edit our `normal_predictive_distribution_fast` from section 3.5 and make it log-normal and dependent on trial:

```
lognormal_model_pred <- function(alpha_samples,
                                 beta_samples,
                                 sigma_samples,
                                 N_obs) {
    # pmap extends map2 (and map) for a list of lists:
    pmap_dfr(list(alpha_samples, beta_samples, sigma_samples),
             function(alpha, beta, sigma) {
                 tibble(
                     trialn = seq_len(N_obs),
                     # we center trial:
                     c_trial = trialn - mean(trialn),
                     # we change the likelihood:
                     # Notice rlnorm and the use of alpha and beta
                     rt_pred = rlnorm(N_obs, alpha + c_trial * beta, sigma)
                 )
             }, .id = "iter") %>%
    # .id is always a string and needs to be converted to a number
        mutate(iter = as.numeric(iter))
}
```

This is our first attempt for a prior predictive distribution:

```
N_obs = 361
alpha_samples <- rnorm(1000, 6, 1.5)
sigma_samples <- rtnorm(1000, 0, 1, a =0)
beta_samples <- rnorm(1000, 0, 1)

prior_pred <- lognormal_model_pred(
```

```
    alpha_samples = alpha_samples,
    beta_samples = beta_samples,
    sigma_samples = sigma_samples,
    N_obs = N_obs)
```

We look here at the median effect:

```
median_effect <-
     prior_pred %>%
     group_by(iter) %>%
     mutate(diff = rt_pred - lag(rt_pred)) %>%
     summarize(
         median_rt = median(diff, na.rm = TRUE)
 )
```

We plot it in Figure 4.4, and as expected is center in zero (as our prior), but we see that the distribution of possible medians for the effect is too spread and includes values that are too extreme.

```
median_effect %>%
    ggplot(aes(median_rt)) +
    geom_histogram()
```

We repeat the same procedure with $\beta \sim Normal(0, .01)$, and we plot it in Figure 4.5. The prior predictive distribution shows us that the prior is still quite vague, it is, howeverm at least in the right order of magnitude. Notice that we are using a distribution of medians because they are less affected by the variance in the posterior predicted distribution; distributions of means will have much more spread. If we want to make the distribution of means more realistic, we would also need to find a more accurate prior for the scale, σ.

Prior selection might look daunting and a lot of work. However, this work is usually done only the first time we encounter an experimental paradigm; besides, priors can be informed by the estimates from previous experiments (even maximum likelihood estimates from frequentist

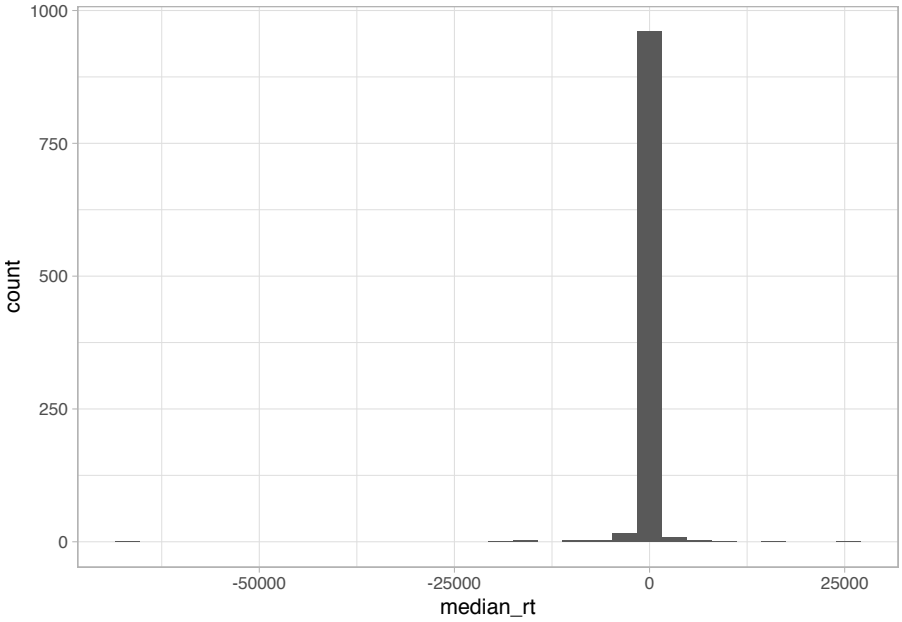


FIGURE 4.4: Prior predictive distribution of the median effect of the model defined in 4.2 with $\beta \sim Normal(0, 1)$.

models can be useful). We will generally use very similar (or identical priors) for analyses dealing with the same type of task. When in doubt, a sensitivity analysis (see section 3.3) can tell us whether the posterior distribution depends unintentionally strongly on our prior selection.

Box 4.2. ***Understanding the Log-normal likelihood***

It is important to understand what we are assuming with our log-normal likelihood. Formally, if a random variable Y is normally distributed with mean μ and variance σ^2, then the transformed random variable $V = \exp(Y)$ is log-normally distributed and has density:

$$LogNormal(v|\mu, \sigma) = f(y) = \frac{1}{\sqrt{2\pi\sigma^2}v} \exp\left(-\frac{(\log(v) - \mu)^2}{2\sigma^2}\right) \tag{4.10}$$

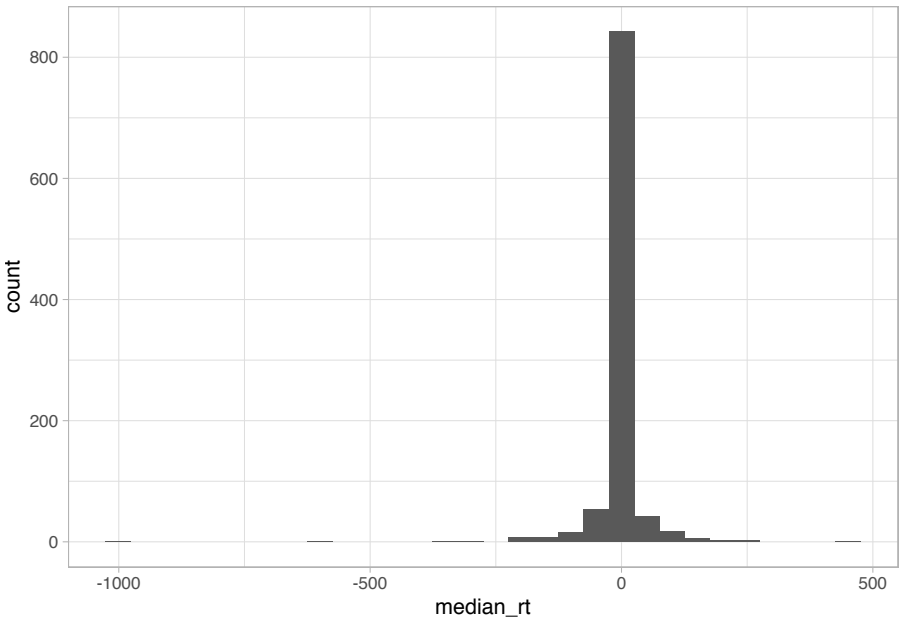


FIGURE 4.5: Prior predictive distribution of the median effect of the model defined in 4.2 with $\beta \sim Normal(0, .01)$.

As explained in section 3.5.2, the model from (4.7) is equivalent to the following:

$$\log(rt_n) \sim Normal(\alpha + c\_trial_n \cdot \beta, \sigma) \tag{4.11}$$

This, in turn, can be written as follows (see ??):

$$\log(rt_n) \sim Normal(\alpha, \sigma) + c\_trial_n \cdot \beta \tag{4.12}$$

We exponentiate both sides, and we use the property of exponents that $\exp(x + y)$ is equivalent to $\exp(x) \cdot exp(y)$.

$$\begin{aligned} rt_n &\sim \exp\left(Normal(\alpha, \sigma) + c\_trial_n \cdot \beta\right) \\ rt_n &\sim \exp\left(Normal(\alpha, \sigma)\right) \cdot \exp\left(c\_trial_n \cdot \beta\right) \\ rt_n &\sim LogNormal(\alpha, \sigma) \cdot \exp\left(c\_trial_n \cdot \beta\right) \end{aligned} \tag{4.13}$$

So, essentially, we are assuming that reaction times are log-normally

distributed with a median of $\exp(\alpha)$ and that the effect of trial number is multiplicative and grows or decays exponentially with the trial number. This has two important consequences:

1. Different values of the intercept, α, given the same β, will affect the difference in reaction times for two adjacent trials (this is contrast to what happens with an additive model such as normal likelihood); see Figure 4.6. This is because, unlike in the additive case, the intercept doesn't cancel out:

   - Additive case:

$$\begin{aligned}&(\alpha + trial_n \cdot \beta) - (\alpha + trial_{n-1} \cdot \beta) = \\ &= \alpha - \alpha + (trial_n - trial_{n-1}) \cdot \beta \\ &= (trial_n - trial_{n-1}) \cdot \beta\end{aligned} \tag{4.14}$$

   - Multiplicative case:

$$\begin{aligned}&\exp(\alpha) \cdot \exp(trial_n \cdot \beta) - \exp(\alpha) \cdot \exp(trial_{n-1} \cdot \beta) = \\ &= \exp(\alpha)(\exp(trial_n \cdot \beta) - \exp(trial_{n-1} \cdot \beta)) \\ &= \exp(\alpha)(\exp(trial_n) - \exp(trial_{n-1}) \cdot \exp(\beta) \\ &\neq (\exp(trial_n) - \exp(trial_{n-1}) \cdot \exp(\beta)\end{aligned} \tag{4.15}$$

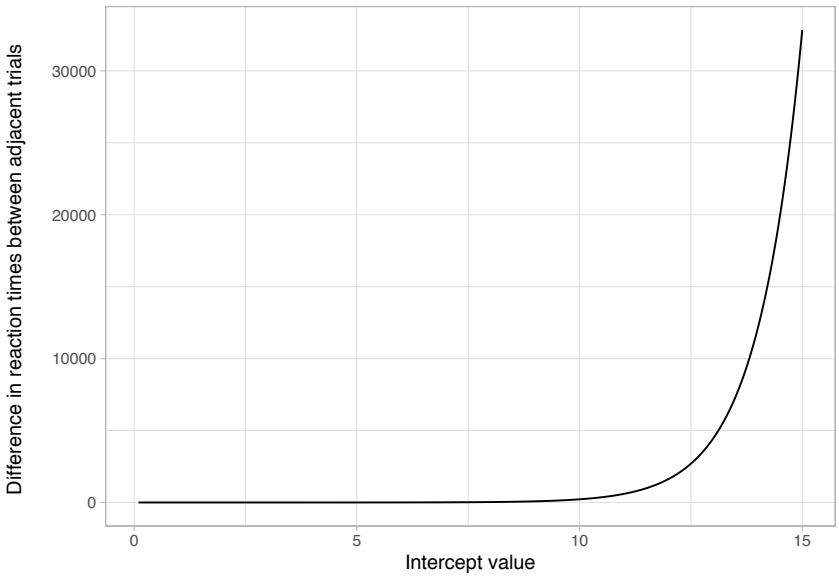


FIGURE 4.6: Fitted value of the difference in reaction time between two adjacent trials, when $\beta = 0.01$ and α lies between 0.1 and 15. The graph shows how changes in the intercept lead to changes in the difference in reaction times between trials, even if β is fixed.

2. As the trial number increases, the same value of β will have a very different impact on the original scale of the dependent variable: $\beta < 0$ will lead to exponential decay and $\beta > 0$ will lead to exponential growth; see figure 4.7.

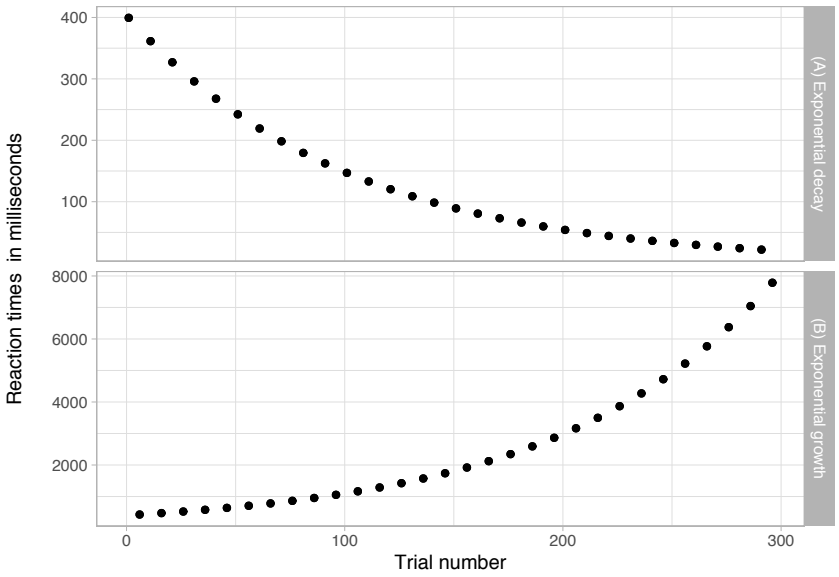


FIGURE 4.7: Fitted value of the dependent variable (reaction times in ms) as function of trial number, when (A) $\beta = -0.01$, exponential decay, and when (B) $\beta = .01$, exponential growth.

Can exponential growth or decay make sense? We need to consider if they do make sense, they will be an approximation valid for a specific range of values, at some point we will expect a ceiling effect: reaction times cannot truly be 0 milliseconds, or take minutes. However, in our specific model, exponential growth or decay *by trial* is probably a bad approximation: We will predict that our participant will take extremely long (if $\beta > 0$) or extremely short (if $\beta < 0$) time in pressing the space bar in a relatively low number of trials. This doesn't mean that the likelihood is wrong by itself, but it does mean that at least we need to put a cap on the growth or decay of our experimental manipulation. We can do this if the exponential growth or decay is a function of, for example, log-transformed trial numbers:

$$rt_n \sim LogNormal(\alpha + \log(c\_trial_n) \cdot \beta, \sigma) \tag{4.16}$$

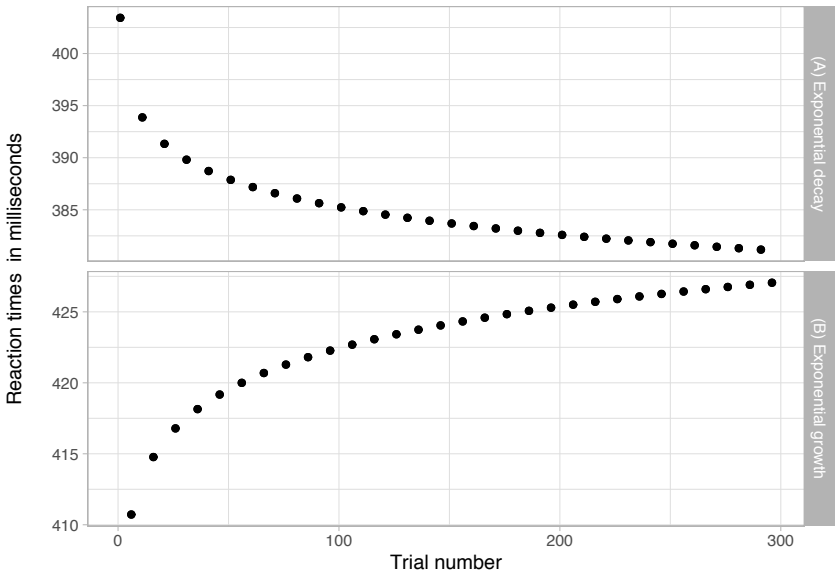


FIGURE 4.8: Fitted value of the dependent variable (reaction times in ms) as function of the natural logarithm of the trial number, when (A) $\beta = -0.01$, exponential decay, and when (B) $\beta = .01$, exponential growth.

Log-normal distributions all the way down

The normal distribution is most often assumed to describe the random variation that occurs in the data from many scientific disciplines. However, most measurements actually show skewed distributions. Limpert et al. (2001) discuss the log-normal distribution in scientific disciplines and how diverse type of data, from lengths of latent periods of infectious diseases to distribution of mineral resources in the Earth's crust, including even body height–the quintessential example of a normal distribution–, closely fit the log-normal distribution.

Limpert et al. (2001) point out that because a random variable that results from multiplicating many independent variables has an approximate log-normal distribution, the most basic indicator of the importance of the log-normal distribution may be very general:

Chemistry and physics are fundamental in life, and the prevailing operation in the laws of these disciplines is multiplication rather than addition.

Furthermore, at many physiological and anatomical levels in the brain, the distribution of numerous parameters is in fact strongly skewed with a heavy tail, suggesting that skewed (typically log-normal) distributions are fundamental to structural and functional brain organization. This might be explained given that the majority of interactions in highly interconnected systems, especially in biological systems, are multiplicative and synergistic rather than additive (Buzsáki and Mizuseki, 2014).

Does the log-normal distribution make sense for reaction times? It has been long noticed that log-normal distribution often provides a good fit to reaction times distributions (**?**). One advantage of assuming log-normally distributed reaction times (but, in fact, this is true for many skewed distributions), is that it entails that the standard deviation of the reaction time distribution will increases with the mean, as has been observed in empirical distributions of reaction times (Wagenmakers et al., 2005). Interestingly, it turns out that log-normal reaction times are also easily generated by certain process models. Ulrich and Miller (1993) shows, for example, that models in which reaction times are determined by a series of processes cascading activation from an input level to an output level (usually passing through a number of intervening processing levels along the way) can generate log-normally distributed reaction times.

4.2.2 The `brms` model

We are now relatively satisfied with the priors for our model, and we can fit the data with `brms`. Notice that we need to specify that the family is `lognormal()`.

```
df_noreading_data <- read_csv("./data/button_press.csv")
df_noreading_data <- df_noreading_data %>%
    mutate(c_trial = trialn - mean(trialn))
```

```
fit_press_trial <- brm(rt ~ 1 + c_trial,
  data = df_noreading_data,
  family = lognormal(),
  prior = c(
    prior(normal(6, 1.5), class = Intercept),
    prior(normal(0, 1), class = sigma),
    prior(normal(0, 1), class = b, coef = c_trial)
  )
)
```

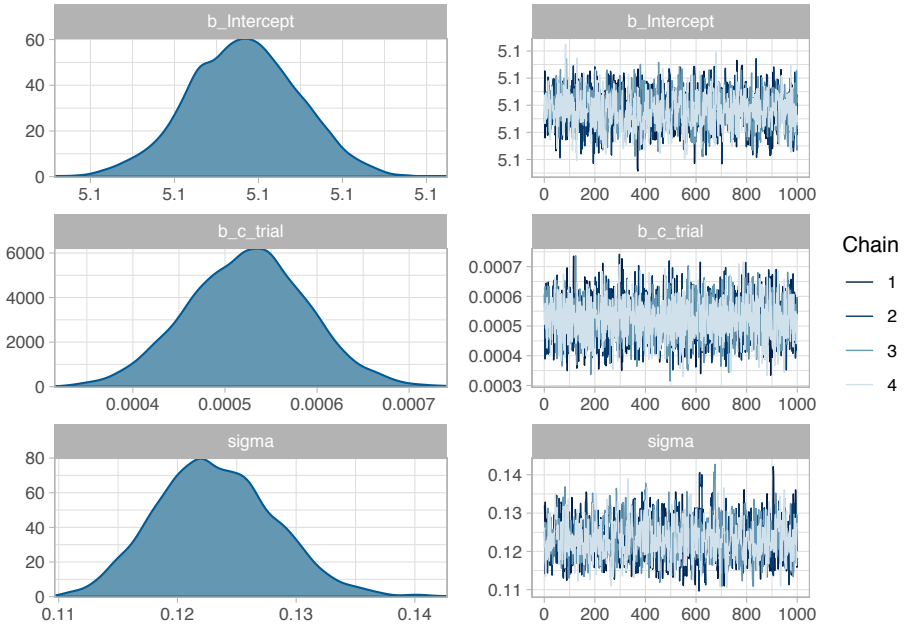
Instead of printing out the complete output from the model, look at the estimates from the posteriors for the parameters α, β, and σ. Notice that these parameters are on the log scale:

```
## print posterior means and 95% credible intervals to four decimal places:
posterior_summary(fit_press_trial)[1:3,c(1,3,4)]
```

```
##             Estimate   Q2.5   Q97.5
## b_Intercept  5.11824 5.1053 5.13100
## b_c_trial    0.00052 0.0004 0.00065
## sigma        0.12323 0.1143 0.13336
```

The posterior distributions can be plotted to obtain a graphical summary of all the parameters in the model:

```
plot(fit_press_trial)
```



Next, we turn to the question of what we can report as our results, and what we can conclude from the data.

4.2.3 How to communicate the results?

As shown above, the first step is to summarize the posteriors in a table or graphically (or both). If the research relates to the effect estimated by the model, the posterior of β can be summarized in the following way: $\hat{\beta} = 0.00052$, 95% CrI = $[0.0004, 0.00065]$.

But in most cases, the effect is easier to interpret in milliseconds. We can transform the estimates back to the millisecond scale from the log scale, but we need to take into account that the scale is not linear, and that the effect between two button presses will differ depending on where we are in the experiment.

We will have a different estimate if we consider the difference between reaction times in a trial at the middle of the experiment (when the centered trial number is zero) and the previous one (when the centered trial number is minus one).

```
alpha_samples<- posterior_samples(fit_press_trial)$b_Intercept
beta_samples<- posterior_samples(fit_press_trial)$b_c_trial
effect_middle_ms <- exp(alpha_samples) - exp(alpha_samples - 1* beta_samples)
## ms effect in the middle of the expt (mean trial vs. mean trial - 1 )
c(mean = mean(effect_middle_ms), quantile(effect_middle_ms, c(.025,.975)))
```

```
##  mean  2.5%   98%
## 0.088 0.066 0.109
```

than if we consider the difference between the second trial and the first one:

```
first_trial <- min(df_noreading_data$c_trial)
second_trial <- min(df_noreading_data$c_trial) +1
effect_beginning_ms <- exp(alpha_samples+  second_trial * beta_samples) -
    exp(alpha_samples+  first_trial * beta_samples)
## ms effect from first to second trial:
c(mean = mean(effect_beginning_ms), quantile(effect_beginning_ms, c(.025,.975)))
```

```
##  mean  2.5%   98%
## 0.080 0.062 0.097
```

There is a slowdown in both cases; when reporting the results of these analyses, one could present the posterior mean and the 95% credible interval and then reason about whether the observed estimates are consistent with the prediction from the theory being investigated.

The practical relevance of the effect for the research question can be important too. For example, only after 100 button presses do we see a slowdown of 9 ms on average ($0.09 \cdot 100$), with a 95% credible interval ranging from 6.62 to 10.88. We need to consider whether our uncertainty of this estimate, and the estimated mean effect have any scientific relevance. Such relevance can be established by considering the previous literature, predictions from a quantitative model, or other expert domain knowledge. Sometimes, a quantitative meta-analysis is helpful; for examples, see Jäger et al. (2017),Mahowald et al. (2016),Nicenboim et al. (2018), and Vasishth et al. (2013). We will discuss concrete examples later in the book, in section ??.

Sometimes, researchers are only interested in establishing that there is an effect; the magnitude and uncertainty of the estimate is of secondary interest. Here, the goal is to argue that there is **evidence** of a slowdown. The word evidence has a special meaning in statistics (Royall, 1997), and in null hypothesis significance testing, a likelihood ratio test is the standard way to argue that one has evidence for an effect. In the Bayesian data analysis context, a Bayes factor hypothesis test must be carried out. We'll come back to this issue in the model comparison section @ref(sec:?).

4.3 Logistic regression: Does set size affect free recall?

We'll look at the capacity limit of working memory to illustrate how the principles we have learned so far can naturally extend to *generalized* linear models (GLMs). In this section, we focus on one special case of GLMS, logistic regression.

For this example, we'll use a subset of the data of Oberauer (2019). We'll focus on one subject who was presented word lists of varying lengths (2, 4, 6, and 8 elements), and then was asked to recall a word given its position on the list; see Figure 4.9.[4]

It is well established that as the number of items to be held in working memory increases, performance, that is accuracy, decreases (among others Oberauer and Kliegl, 2001). We will investigate whether we can establish this finding with data from only one subject.

```
df_recall_data <- read_table2("./data/PairsRSS1_all.dat",
                              col_names = c("subject", "session", "block",
                                            "trial", "set_size",
                                            "response_size_list",
                                            "response_size_new_words",
                                            "tested", "response",
```

[4] We will only use data from the recall test in which the participant had to type the probed word (and we will ignore the trials with multiple forced choice for ease of explanation).

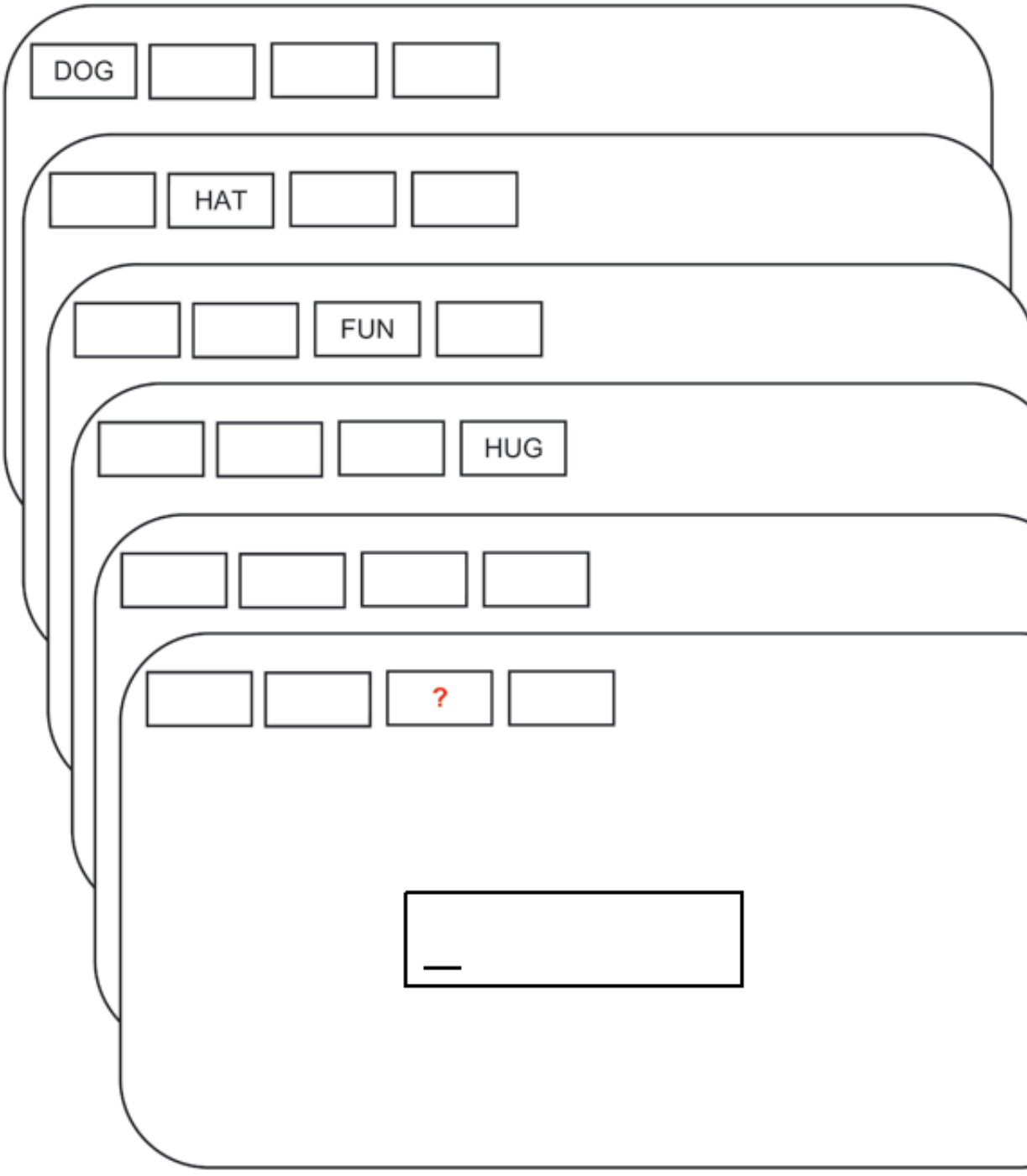


FIGURE 4.9: Flow of events in a trial with memory set size 4 and free recall. Adapted from Oberauer (2019); licensed under CC BY 4.0.

```
                                                "response_category", "rt")) %>%
    # We ignore the type of incorrect responses (the focus of the paper)
    mutate(correct = if_else(response_category ==1, 1, 0)) %>%
    # and we only use the data from the free recall task:
    # (when there was no list of possible responses)
    filter(response_size_list + response_size_new_words == 0) %>%
    # We select one subject
    filter(subject == 10) %>%
    mutate(c_set_size = set_size - mean(set_size))
# we can ignore the warning from read_table

# Set sizes in the dataset:
df_recall_data$set_size %>%
    unique
```

```
## [1] 4 8 2 6
```

```
# Trials by set size
df_recall_data %>%
    group_by(set_size) %>%
    count()
```

```
## # A tibble: 4 x 2
## # Groups:   set_size [4]
##   set_size     n
##      <dbl> <int>
## 1        2    23
## 2        4    23
## 3        6    23
## 4        8    23
```

The data look like this: the column `correct` records the 0 (incorrect) or 1 (correct) responses, and the column `c\_set\_size` records the centered memory set size; these latter scores have continuous values -3, -1, 1, and 3. These continuous values are centered versions of 2, 4, 6, and 8.

```
df_recall_data<-df_recall_data[,c(12,13)]
df_recall_data
```

```
## # A tibble: 92 x 2
##   correct c_set_size
##     <dbl>      <dbl>
## 1       1         -1
## 2       0          3
## 3       1         -3
## 4       1          1
## 5       1         -1
## # ... with 87 more rows
```

We want to model the trial by trial accuracy and examine whether the probability of recalling a word is related to the number of words in the set that the subject needs to remember.

4.3.1 The likelihood for the logistic regression model

Recall that the Bernoulli likelihood generates a 0 or 1 response with a particular probability θ. For example, one can generate simulated data for 10 trials as follows:

```
as.numeric(rbernoulli(n=10,p=0.5))
```

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

We can therefore define each dependent value correct_n in the data as being generated from a Bernoulli random variable with probability of success θ_n. Here, $n = 1, \ldots, N$ indexes the trial, correct_n is the dependent variable (0 indicates an incorrect recall and 1 a correct recall), and θ_n is the probability of correctly recalling a probe in a given trial n.

$$correct_n \sim Bernoulli(\theta_n) \tag{4.17}$$

Since θ_n is bounded to be between 0 and 1 (it is a probability), we cannot just fit a regression model using the normal or lognormal likelihood as we did in the preceding examples. Such a model would be inappropriate

because it would assume that the data range from $-\infty$ to $+\infty$, but in fact the probabilities we want to estimate lie within the [0,1] boundary.

The generalized linear modeling framework solves this problem by defining a so-called **link function** $g(\cdot)$ that connects the linear model to the quantity to be estimated (here, the probabilities θ_n). The link function used for 0,1 responses is called the **logit link**, and is defined as follows.

$$\eta_n = g(\theta_n) = \log\left(\frac{\theta_n}{1-\theta_n}\right) \tag{4.18}$$

The term $\frac{\theta_n}{1-\theta_n}$ is called the **odds**.[5] The logit link function is therefore a log-odds; it maps real numbers ranging from $(-\infty, +\infty)$ to probability values ranging from $[0, 1]$. Figure 4.10 shows the logit link function, $\eta = g(\theta)$, and the inverse logit, $\theta = g^{-1}(\eta)$, which is called the **logistic function**; the relevance of this logistic function will become clear in a moment.

The linear model is now fit not to the 0,1 responses as the dependent variable, but to η_n, i.e., log-odds, as the dependent variable:

$$\eta_n = \log\left(\frac{\theta_n}{1-\theta_n}\right) = \alpha + \beta \cdot c\_set\_size \tag{4.19}$$

Once η_n is estimated, one can easily compute the parameters of interest, the estimated probabilities, by solving the above equation for θ_n (in other words, by computing the inverse of the logit function), which is the above-mentioned logistic regression function:

$$\theta_n = g^{-1}(\eta_n) = \log\left(\frac{\exp(\eta_n)}{1+\exp(\eta_n)}\right) \tag{4.20}$$

In summary, the generalized linear model with the logit link fits the following Bernoulli likelihood:

[5] Odds are defined to be the ratio of the probability of success to the probability of failure. For example, the odds of obtaining a one in a fair six-sided die are $\frac{1/6}{1-1/6} = 1/5$. The odds of obtaining a heads in a fair coin are 1/1.

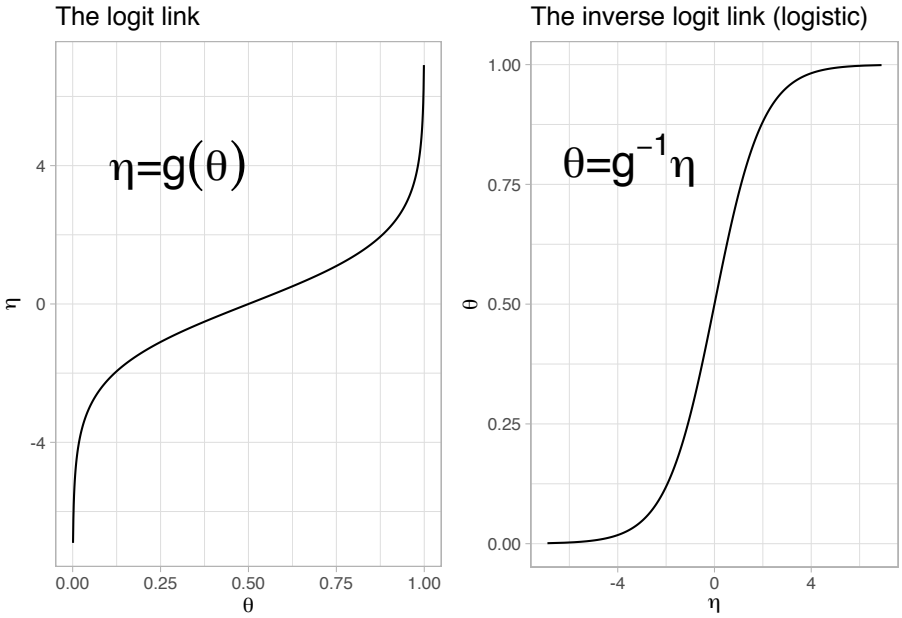


FIGURE 4.10: The logit and inverse logit (logistic) function.

$$correct_n \sim Bernoulli(\theta_n) \tag{4.21}$$

The model is fit on the log-odds scale, $\eta_n = \alpha + c\_set\_size_n \cdot \beta$. Once η_n has been estimated, the inverse logit or the logistic function is used to compute the probability estimates $\theta_n = \log(\frac{\exp(\eta_n)}{1+\exp(\eta_n)})$‘. An example of the calculations will be shown in the next section.

4.3.2 Priors for logistic regression

In order to decide on priors for α and β we need to take into account that these parameter do not represent probabilities or proportions, but *log-odds*, the x-axis in Figure 4.10 (right-hand side figure). As shown in the figure, the relationship between log-odds and probabilities is not linear.

There are two functions in R that implement the logit and inverse logit functions: `qlogis(p)` for the logit function and `plogis(x)` for the inverse logit or logistic function.

Now we need to set priors for α and β. Given that we centered our pre-

dictor, the intercept, α, represents the probability of correctly recalling one word in a random position for the average set size of five (since $5 = \frac{2+4+6+8}{4}$), which, incidentally, was not presented in the experiment. This is one case where the intercept doesn't have a clear interpretation if we leave the prediction uncentered: With non-centered set size, the intercept will be the probability of recalling one word in a set of *zero* words.

The prior for α will depend on how difficult the recall task is. If we are not sure, we could assume that the probability of recalling a word for an average set size, α, is centered in .5 (a 50/50 chance) with a great deal of uncertainty. The `R` command `plogis(.5)` tells us that .5 corresponds to zero in log-odds. How do we include a great deal of uncertainty? We could look at Figure 4.10, and decide on a standard deviation of 4 in a normal distribution centered in zero:

$$\alpha \sim Normal(0, 4) \tag{4.22}$$

Let's plot this prior in log-odds and in probability scale by drawing random samples.

```
samples_logodds <- tibble(alpha = rnorm(100000, 0, 4))
samples_prob <- tibble(p = plogis(rnorm(100000, 0, 4)))
ggplot(samples_logodds, aes(alpha)) +
    geom_density()
ggplot(samples_prob, aes(p)) +
    geom_density()
```

Figure 4.11 shows that our prior assigns more probability mass to extreme probabilities of recall than to intermediate values. Clearly, this is not what we intended.

We could try several values for standard deviation of the prior, until we find a prior that make sense for us. Reducing the standard deviation to 1.5 seems to make sense as shown in Figure 4.12.

$$\alpha \sim Normal(0, 1.5) \tag{4.23}$$

We need to decide now on the prior for the effect in log-odds of increasing

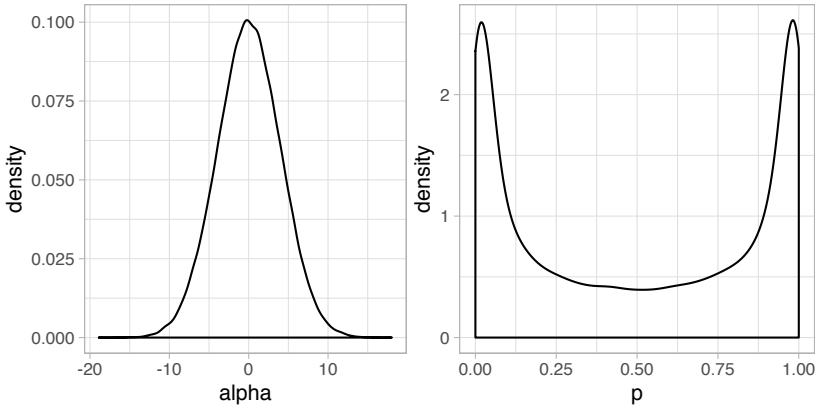


FIGURE 4.11: Prior for $\alpha \sim Normal(0, 4)$ in log-odds and in probability space.

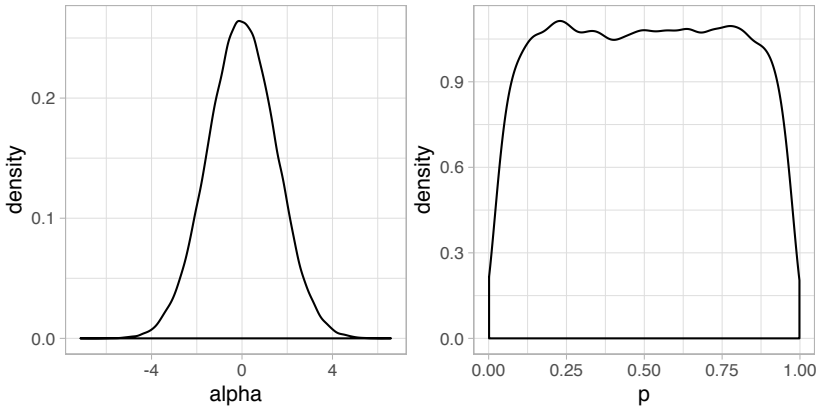


FIGURE 4.12: Prior for $\alpha \sim Normal(0, 4)$ in log-odds and in probability space.

the set size, β. We are going to choose a normal distribution centered on zero, reflecting our lack of any commitment regarding the direction of the effect. Let's get some intuitions regarding different possible standard deviations for this prior, by testing the following distributions as priors:

(a) $\beta \sim Normal(0, 1)$
(b) $\beta \sim Normal(0, .5)$
(c) $\beta \sim Normal(0, .1)$
(d) $\beta \sim Normal(0, .01)$

(e) $\beta \sim Normal(0, .001)$

The following function is an edited version of the earlier `normal_predictive_distribution_fast` from section 3.5; it has been edited to make it compatible with logistic regression and dependent on set size:

```
logistic_model_pred <- function(alpha_samples,
                                beta_samples,
                                set_size,
                                 N_obs) {
    map2_dfr(alpha_samples, beta_samples,
             function(alpha, beta) {
                 tibble(
                     set_size = set_size,
                     # we center size:
                     c_set_size = set_size - mean(set_size),
                     # change the likelihood:
                     # Notice the use of a link function for alpha and beta
                     theta = plogis(alpha + c_set_size * beta),
                     # There is no bernoulli in R, but we can just use
                     # binomial when the total number of trials is 1
                     correct_pred = rbinom(N_obs, size = 1, prob = theta)
                 )
             }, .id = "iter") %>%
    # .id is always a string and needs to be converted to a number
        mutate(iter = as.numeric(iter))
}
```

Let's assume 800 observations with 200 observation of each set size:

```
N_obs <- 800
set_size <- rep(c(2,4,6,8),200)
```

We iterate over the four possible standard deviations of β:

```
alpha_samples <- rnorm(1000, 0, 1.5)
sds_beta <- c(1, 0.5, 0.1,0.01, 0.001)
prior_pred <- map_dfr(sds_beta, function(sd) {
    beta_samples <- rnorm(1000, 0, sd)
    logistic_model_pred(alpha_samples = alpha_samples,
                        beta_samples = beta_samples,
                        set_size = set_size,
                        N_obs = N_obs
                        ) %>%
        mutate(prior_beta_sd = sd)
})
```

And we calculate the accuracy for each one of the priors we want to examine, for each iteration, and for each set size.

```
mean_accuracy <-
     prior_pred %>%
     group_by(prior_beta_sd, iter, set_size) %>%
     summarize(accuracy = mean(correct_pred)) %>%
     mutate(prior = paste0("Normal(0, ",prior_beta_sd,")"))
```

We plot it in Figure 4.13, and as expected the priors are centered at zero. We see that the distribution of possible accuracies for the prior that has a standard deviation of one is problematic: There is too much probability mass concentrated near zero and one for set sizes of 2 and 8.

```
mean_accuracy %>%
    ggplot(aes(accuracy)) +
    geom_histogram() +
    facet_grid(set_size~prior)
```

It's hard to tell the differences between the other priors, and it might be more useful to look at the predicted differences in accuracy between set sizes. We calculate them as follows:

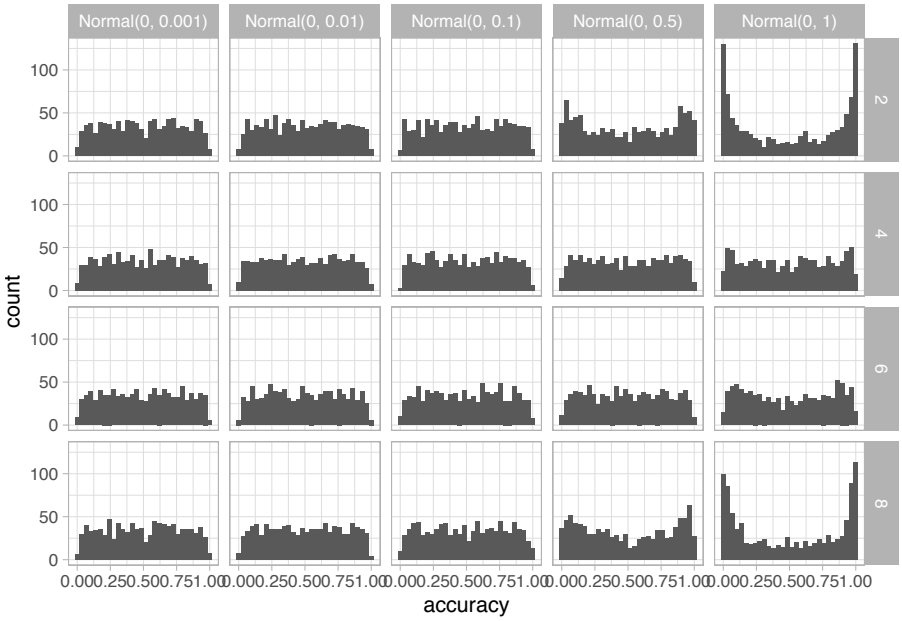


FIGURE 4.13: Prior predictive distribution of mean accuracy of the model defined in 4.3, for different set sizes and different priors for β.

```
diff_accuracy <- mean_accuracy %>%
    arrange(set_size) %>%
    group_by(iter, prior_beta_sd) %>%
    mutate(diffaccuracy = accuracy - lag(accuracy) ) %>%
    mutate(diffsize = paste(set_size,"-",  lag(set_size))) %>%
    filter(set_size >2)
```

We plot them in Figure 4.14. If we are not sure whether the increase of set size could produce something between a null effect and a relatively large effect, we can choose the prior with a standard deviation of 0.5.

```
diff_accuracy %>%
    ggplot(aes(diffaccuracy)) +
    geom_histogram() +
    facet_grid(diffsize~prior)
```

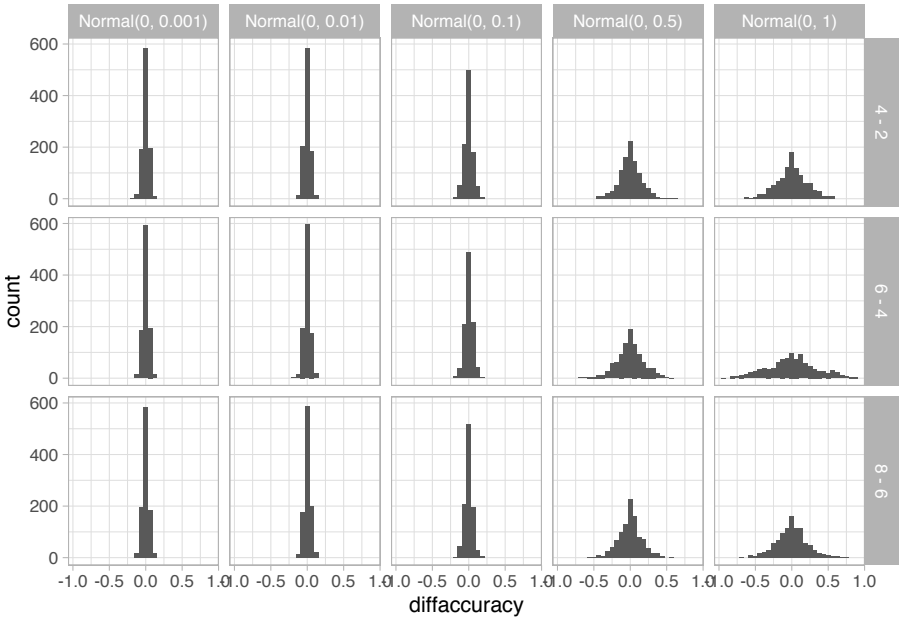


FIGURE 4.14: Prior predictive distribution of differences in mean accuracy between set sizes of the model defined in 4.3 for different priors for β.

We settle one the following priors:

$$\begin{aligned} \alpha &\sim Normal(0, 1.5) \\ \beta &\sim Normal(0, 0.5) \end{aligned} \tag{4.24}$$

4.3.3 The `brms` model

Having decided on the likelihood, the link function, and the priors, the model can now be fit using `brms`. Notice that we need to specify that the family is `bernoulli()`, and the link is `logit`.

```
df_recall_data <- read_table2("./data/PairsRSS1_all.dat",
                              col_names = c("subject", "session", "block",
                                            "trial", "set_size",
                                            "response_size_list",
                                            "response_size_new_words",
```

```
                                            "tested", "response",
                                            "response_category", "rt")) %>%
    # ignore the type of incorrect responses (the focus of the paper)
    mutate(correct = if_else(response_category ==1, 1, 0)) %>%
    # use only the data from the free recall task:
    # (when there was no list of possible responses)
    filter(response_size_list + response_size_new_words == 0) %>%
    # select one subject
    filter(subject == 10) %>%
    mutate(c_set_size = set_size - mean(set_size))
# ignore the warning from read_table

# Set sizes in the dataset:
df_recall_data$set_size %>%
    unique
```

```
## [1] 4 8 2 6
```

```
# Trials by set size
df_recall_data %>%
    group_by(set_size) %>%
    count()
```

```
## # A tibble: 4 x 2
## # Groups:   set_size [4]
##   set_size     n
##      <dbl> <int>
## 1        2    23
## 2        4    23
## 3        6    23
## 4        8    23
```

```
fit_recall <- brm(correct ~ 1 + c_set_size,
  data = df_recall_data,
  family = bernoulli(link = logit),
```

```
  prior = c(
    prior(normal(0, 1.5), class = Intercept),
    prior(normal(0, .5), class = b, coef = c_set_size)
  )
)
```

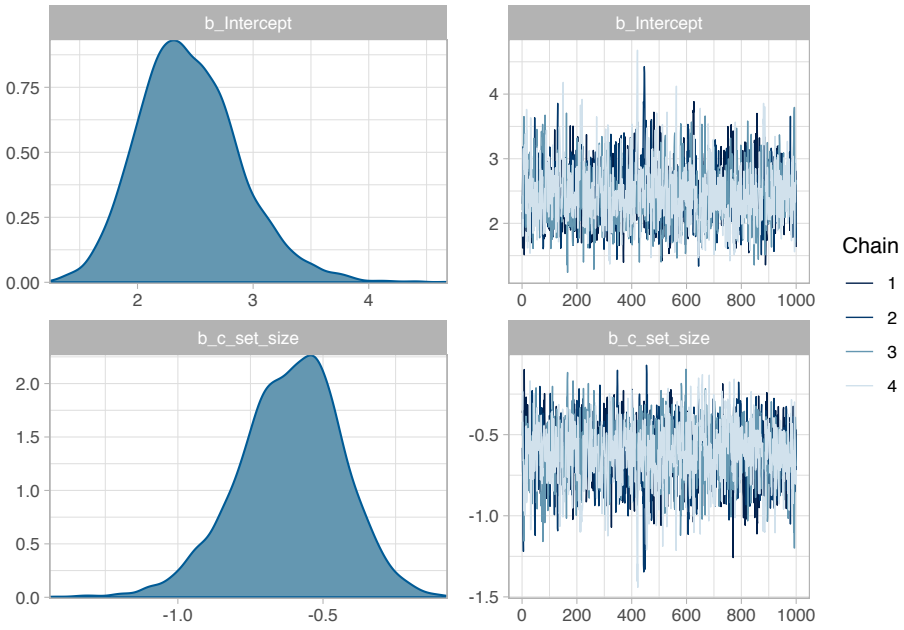
Next, look at the summary of the posteriors of each of the parameters. Keep in mind that the parameters are in log-odds space:

```
posterior_summary(fit_recall)[1:2,c(1,3,4)]
```

```
##               Estimate  Q2.5 Q97.5
## b_Intercept       2.45  1.70  3.40
## b_c_set_size     -0.61 -0.99 -0.29
```

Plot the posteriors as well:

```
plot(fit_recall)
```



Next, we turn to the question of what we can report as our results, and what we can conclude from the data.

4.3.4 How to communicate the results?

We are here in situation analogous as before with the log-normal model. If we want to talk about the effect estimated by the model in log-odds space, we summarize the posterior of β in the following way: $\hat{\beta} = -0.61$, 95% CrI = $[-0.99, -0.29]$.

However, the effect might be easier to understand in proportions rather than in log-odds. Let's look at the average accuracy for the task first:

```
alpha_samples<- posterior_samples(fit_recall)$b_Intercept
av_accuracy <- plogis(alpha_samples)
c(mean = mean(av_accuracy), quantile(av_accuracy, c(.025,.975)))
```

```
## mean 2.5%  98%
## 0.92 0.85 0.97
```

As before, to transform the effect of our manipulation to an easier to interpret scale (i.e., proportion), we need to take into account that the scale is not linear, and that the effect of increasing the set size depends on the average accuracy, and the set size that we start from.

We can do the following calculation, similar to what we did for the trial effects experiment, to find out the decrease in accuracy in proportions or probability scale:

```
beta_samples<- posterior_samples(fit_recall)$b_c_set_size
effect_middle <- plogis(alpha_samples) - plogis(alpha_samples - beta_samples)
c(mean = mean(effect_middle), quantile(effect_middle, c(.025,.975)))
```

```
##   mean   2.5%    98%
## -0.034 -0.055 -0.017
```

Notice the interpretation here, if we increase the set size from the average set size minus one to the average set size, we get a reduction in the accuracy of recall of -0.03, 95% CrI = $[-0.06, -0.02]$. Recall that the av-

erage set size, 5, was not presented to the subject! We could also look at the decrease in accuracy from a set size of 2 to 4:

```
effect_4m2 <- plogis(alpha_samples+  (4 - mean(df_recall_data$set_size)) * beta_samples) -
    plogis(alpha_samples+  (2 - mean(df_recall_data$set_size)) * beta_samples)
c(mean = mean(effect_4m2), quantile(effect_4m2, c(.025,.975)))
```

```
##   mean   2.5%    98%
## -0.032 -0.057 -0.011
```

We see that increasing the set size does have a detrimental effect in recall, as we suspected.

4.3.5 Descriptive adequacy

One potentially useful aspect of posterior distributions is that we could also make predictions for other conditions not presented in the actual experiment, such as set sizes that weren't tested. We could then verify if our model was right with another experiment. To make predictions for other set sizes, we extend our dataset adding rows with set sizes of 3, 5, and 7. To be consistent with the data of the other set sizes in the experiment, we add 23 trials of each new set size (this is the number of trial by set sizes in the dataset). Something important to notice is that **we need to center our predictor based on the original mean set size**. This is because we want to maintain our interpretation of the intercept. We extend the data as follows, and we summarize the data and plot it in Figure 4.15.

```
df_recall_data_ext <- df_recall_data %>%
    bind_rows(tibble(set_size = rep(c(3,5,7),23),
                     c_set_size = set_size - mean(df_recall_data$set_size)))
df_recall_pred_ext <- posterior_predict(fit_recall,
                                        newdata = df_recall_data_ext,
                                        nsamples = 1000) %>%
    array_branch(margin = 1) %>%
    map_dfr( function(yrep_iter) {
        df_recall_data_ext %>%
            mutate(correct = yrep_iter)
```

```
    }, .id = "iter") %>%
    mutate(iter = as.numeric(iter))
```

```
df_recall_pred_ext_summary <- df_recall_pred_ext %>%
    group_by(iter, set_size) %>%
    summarize(accuracy = mean(correct))
# observed means:
df_recall_summary<- df_recall_data %>%
    group_by(set_size) %>%
    summarize(accuracy = mean(correct))
ggplot(df_recall_pred_ext_summary, aes(accuracy)) +
    geom_histogram(alpha=.5)+
    geom_vline(aes(xintercept= accuracy),data= df_recall_summary)+
    facet_grid(set_size ~ .)
```

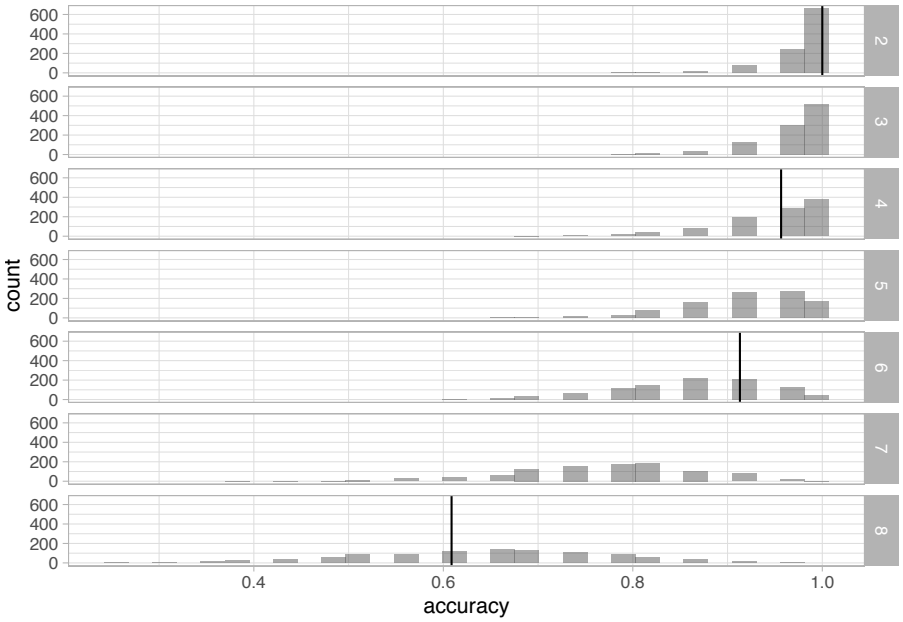


FIGURE 4.15: Distribution of posterior predicted mean accuracies in gray for tested set sizes (2, 4, 6, and 8) and untested ones (3, 5, and 7), and observed mean accuracy in black lines by tested set sizes.

4.4 Summary

4.5 Further reading

4.6 Exercises

1. For the lognormal model (section 4.2:
    1. Estimate the slowdown in milliseconds for the last time the subject pressed the space bar in the experiment. In addition, predict the slowdown if the experiment would have had 500 observations. Does it work? What does it tell us about our covariate?

4.7 Appendix

4.7.1 Preparation of the pupil size data (section 4.1)

We prepared the data for the linear model of section 4.1 with a dataset from Wahn et al. (2016) downloaded from (https://osf.io/z43dz/). The following code creates the pilot data and the averages by trial.

```
df_pupil_full_control <- read_csv("./data/PLOS_raw_pupildata_controlexperiment.csv") %>%
    rename(p_size = Pupilsize, load = Attentionalload, trial = Trial, time = Time)
# "Pilot" data
df_pupil_pilot <- df_pupil_full_control %>%
    filter(Subnum==701, load ==0,time <100, trial ==5) %>%
    select(time, p_size, load)
write_csv(df_pupil_pilot, "./data/pupil_pilot.csv")
# Averaged by trial data
df_pupil <- df_pupil_full_control %>%
    filter(Subnum==701, time > 100) %>%
    group_by(trial, load) %>%
```

```
    summarize(p_size = mean(p_size, na.rm= TRUE))
write_csv(df_pupil, "./data/pupil.csv")
```

5

Bayesian hierarchical models

Usually, experimental data in cognitive science contain "clusters". These are natural groups that contain observations that are more similar within the clusters than between them. The most common examples of clusters in experimental designs are participants and items. These clusters arise because we have multiple (repeated) observations for each participant, and for each item. If we want to incorporate this structure in our analysis, we need to use a hierarchical model (also called multi-level and mixed models).

The notion of exchangeability is important to understand in hierarchical models. Suppose we assign a numerical index to each of the levels of a cluster (e.g., to each participant). When the levels are exchangeable, we can reassign the indices arbitrarily and lose no information. In hierarchical models, we treat the clusters as exchangeable, and observations within each cluster as also exchangeable. In practice, we include predictors at the level of the observations, those are the predictors that correspond to the experimental manipulations (e.g., attentional load, trial number, Cloze probability, etc.); and maybe also at the cluster level, these are predictors that indicate characteristics of the cluster (e.g., working memory capacity score of each participant). Then the conditional distributions given these explanatory variables would be exchangeable, that is, our predictors incorporate all the information that is not exchangeable, and when we factor the predictors out, the observations or clusters are exchangeable. This is the reason why while the item number is an appropriate cluster (the indexes are exchangeable), trial number is not. See Chapter 5 of Gelman et al. (2014) for a more technical treatment of exchangeability.

to-do: make this better

5.1 A hierarchical normal model: The N400 effect

Event-related potentials (ERPs) allow scientists to observe electrophysiological responses in the brain measured by means of electroencephalography (EEG) that are time-locked to a specific event (i.e., the presentation of the stimuli). A very robust ERP effect in the study of language is the so-called N400. It has been shown that words with low predictability are accompanied by an *N400 effect* in comparison with high-predictable words, this is a relative negativity that peaks around 300-500 after word onset over central parietal scalp sites (first noticed in Kutas and Hillyard, 1980; for semantic anomalies and in Kutas and Hillyard, 1984, for low predictable word; for a review: Kutas and Federmeier, 2011).

to-do: N400 plot here

In 1, for example, the continuation *'paint'* has higher predictability than the continuation *'dog'*, and thus we would expect a more negative signal, that is, an N400 effect, in *'dog'* in (b) in comparison with *'paint'* in (a). It is often the case that predictability is measured with a Cloze task (see section 1.3).

1. Example from Kutas and Hillyard (1984)
   a. Don't touch the wet paint.
   b. Don't touch the wet dog.

The EEG data are typically recorded in tens of electrodes every couple of milliseconds, but for our purposes (i.e., for learning about Bayesian hierarchical models), we can safely ignore the complexity of the data. A common way to simplify the high-dimensional EEG data when we are dealing with the N400 is to focus on the average amplitude of the EEG signal at its

typical spatio-temporal window of the N400 (see for example Frank et al., 2015).

For this example, we are going to focus on the N400 effect for critical nouns from a subset of the data of Nieuwland et al. (2018). Nieuwland et al. (2018) presented a replication attempt of an original experiment of DeLong et al. (2005) with sentences like (2)

2. Example from DeLong et al. (2005)
   a. The day was breezy so the boy went outside to fly a kite.
   b. The day was breezy so the boy went outside to fly an airplane.

We'll ignore the goal of original experiment (DeLong et al., 2005), and its replication (Nieuwland et al., 2018). We are going to focus on the N400 at the final nouns in the experimental stimuli. In example (2), for example, the final noun *'kite'* has higher predictability than *'airplane'*, and thus we would expect a more negative signal in *'airplane'* in (b) in comparison with *'kite'* in (a).

To speed-up computation, we'll restrict the dataset to the participants from the Edinburgh lab; the entire dataset can be found in `https://osf.io/q7dsk/`.

```
df_eeg_data <- read_tsv("data/public_noun_data.txt") %>%
    filter(lab=="edin")
df_eeg_data
```

```
## # A tibble: 2,827 x 6
##   subject segment cloze lab    item  n400
##   <chr>     <dbl> <dbl> <chr> <dbl> <dbl>
## 1 edin1         1     0 edin    101  7.08
## 2 edin1         2     3 edin    102 -0.68
## 3 edin1         3   100 edin    103  1.39
## 4 edin1         4    93 edin    104 22.8
## 5 edin1         5     0 edin    105  1.61
## # ... with 2,822 more rows
```

```
# Number of subjects
df_eeg_data %>%
    distinct(subject) %>%
    count()
```

```
## # A tibble: 1 x 1
##       n
##   <int>
## 1    37
```

```
## choose only the relevant columns:
df_eeg_data<-df_eeg_data[,c(1,3,5,6)]
## strip edin prefix on subject names:
df_eeg_data$subject<-stringr::str_replace(df_eeg_data$subject,"edin","")
## save unique subject ids:
subj_levels<-sort(unique(as.numeric(as.character(df_eeg_data$subject))))
df_eeg_data$subject<-factor(df_eeg_data$subject,levels=subj_levels)
```

In the data, the Cloze is in percentages, we'll transform it to proportions and center it before using it as a predictor in `c_cloze`.

```
df_eeg_data  <- df_eeg_data %>%
    mutate(c_cloze= cloze/100 - mean(cloze/100) )
df_eeg_data$c_cloze %>% summary()
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   -0.47   -0.44    0.03    0.00    0.43    0.53
```

One nice aspect of using averages of EEG data is that they are roughly normally distributed. This allows us to use the Normal likelihood. Here is a histogram showing the distribution of the data:

```
df_eeg_data %>% ggplot(aes(n400)) +
    geom_histogram(binwidth = 4, colour="gray", alpha = .5, aes(y = ..density..)) +
    stat_function(fun = dnorm, args = list(
                                            mean = mean(df_eeg_data$n400),
```

```
                                                    sd = sd(df_eeg_data$n400))) +
    xlab("Average voltage in microvolts for the N400 spatiotemporal window")
```

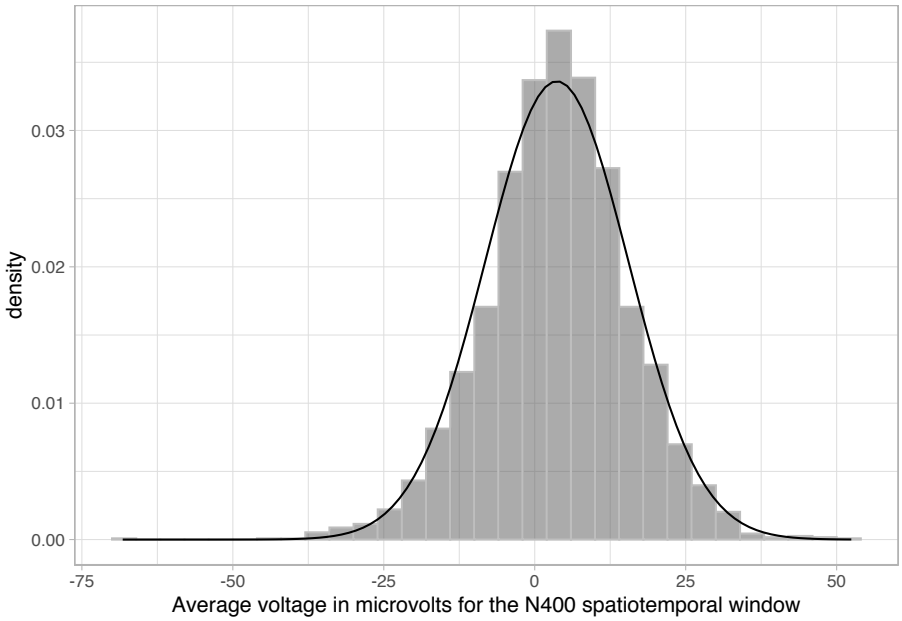


FIGURE 5.1: Histogram of the N400 averages for every trial in gray; density plot of a normal distribution in black.

5.1.1 Complete-pooling model (M_{cp})

We'll start from the simplest model which is basically the linear regression we encountered in the preceding chapter.

5.1.1.1 Model assumptions

This model, call it M_{cp}, makes the following assumptions.

1. The EEG averages for the N400 spatiotemporal window are normally distributed.
2. Observations are *independent*.
3. There is a linear relationship between cloze and the EEG signal for the trial.

Note that this model is incorrect for these data due to assumption 2 being violated.

With the last assumption, we are saying that the difference in the average signal when we compare nouns with Cloze probability 0-.1, .1-.2, .2-.3, and so forth is the same. Notice that this *is* an assumption, and it may not necessarily be the case in the actual data. This means that we are going to get a posterior for β *conditional* on the assumption that the linear relationship holds. Even if it *approximately* holds, we still don't know how much we deviate from this assumption. We'll come back to this issue in chapter ?? when we deal with model comparison.

We can now decide on a likelihood and priors:

5.1.1.2 Likelihood and priors

A normal likelihood seems reasonable for these data:

$$signal_n \sim Normal(\alpha + c\_cloze_n \cdot \beta, \sigma) \tag{5.1}$$

where $n = 1, \dots, N$, and rt is the dependent variable (reaction times in milliseconds). The variable N represents the total number of data points.

As always we need to rely on our previous knowledge and domain expertise to decide on priors. We know that ERPs (signals time-locked to a stimulus) have mean amplitudes of a couple of microvolts, this is easy to see in any plot of the EEG literature. This means that we don't expect the effect of our manipulation to exceed, say, 10 μV. As before we'll assume that effects can be negative or positive. We can quantify our prior knowledge regarding plausible values of β as a normal distributed centered at zero with a standard deviation of 10 μV.

If the signal for each ERP is *baselined*, that is, the mean signal of a time window before the time window of interest is subtracted to the time window of interest, then the mean signal would be relatively close to 0. Since we know the ERPs were baselined in this study, we expect that the grand mean of our signal should be relatively close to zero. Our prior for α is then normally distributed centered in zero with a standard deviation of 10 μV.

The standard deviation of our signal distribution is harder to guess. We

know that EEG signals are quite noisy, and that the standard deviation must be higher than zero. Our prior for σ is a truncated normal distribution with location zero and scale as 50. Notice that since we truncate the distribution, the parameters location and scale do not correspond to mean and standard deviation; we can see this by drawing random samples from this distribution and calculating their mean and standard deviation:

```
samples <- rtnorm(1000,0 ,50, a=0)
c(mean = mean(samples), sd(samples))
```

```
## mean
##   40   31
```

So we are essentially saying that we assume a prior that we will find the true standard deviation of the signal in the following interval with 95% probability:

```
quantile(samples, c(0.025,.975))
```

```
##  2.5%   98%
##   1.7 111.1
```

To sum up, we are going to use the following Priors:

$$
\begin{aligned}
\alpha &\sim Normal(0, 10) \\
\beta &\sim Normal(0, 10) \\
\sigma &\sim Normal_{+}(0, 50)
\end{aligned}
\tag{5.2}
$$

A model such as M_{cp} is sometimes called a *fixed-effects* model: all the parameters are fixed and do not vary from subject to subject or from item to item. A similar frequentist model would correspond to fitting a simple linear model using the `lm` function: `lm(n400 ~ 1 + cloze, data=df_eeg_data)`.

We fit this model in `brms` as follows (the default family is `gaussian()` so we can omit it). As with `lm`, by default an intercept is fitted and thus `n400 ~ c_cloze` is equivalent to `n400 ~ 1 + c_cloze`:

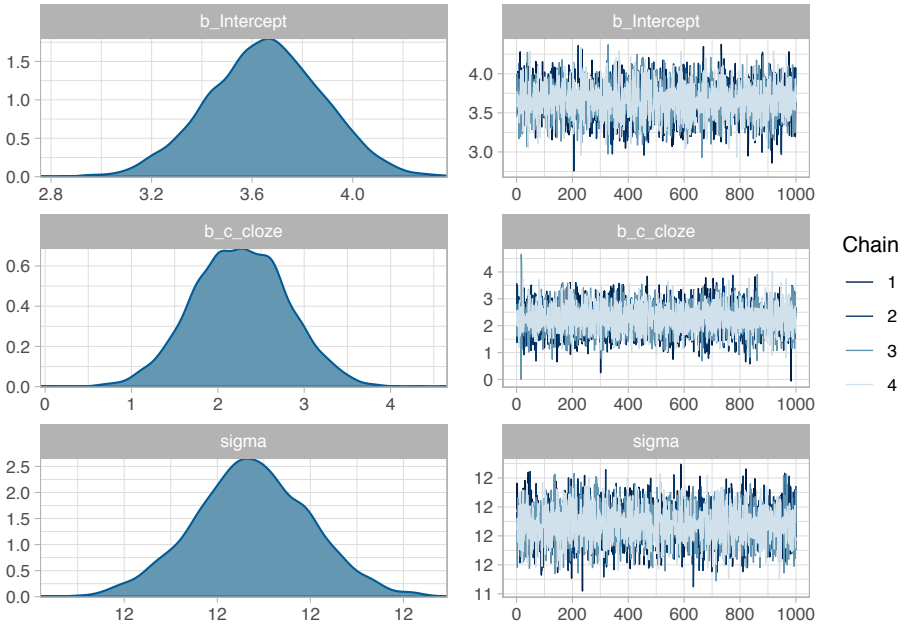
```
fit_N400_cp <- brm(n400 ~ c_cloze,
  prior =
    c(prior(normal(0, 10), class = Intercept),
      prior(normal(0, 10), class = b, coef = c_cloze),
      prior(normal(0, 50), class = sigma)),
  data = df_eeg_data
)
```

For now, we'll check the summary and plot the posterior of the model. To save space we use the function `posterior_summary()`, but it's a good idea to inspect the full summary.

```
posterior_summary(fit_N400_cp)
```

```
##             Estimate Est.Error     Q2.5    Q97.5
## b_Intercept      3.7      0.23      3.2      4.1
## b_c_cloze        2.3      0.54      1.2      3.3
## sigma           11.8      0.15     11.5     12.1
## lp__        -11005.9      1.21 -11009.0 -11004.5
```

```
plot(fit_N400_cp)
```



5.1.2 No-pooling model (M_{np})

One of the assumptions of the previous model is clearly wrong: observations are not independent, they are clustered the participant (and also on the specific item, but we'll ignore this until section 5.1.4). It is reasonable to assume that EEG signals are more similar within participants than between them. The following model assumes that each participant is completely independent from each other.[1]

5.1.2.1 Model assumptions

1. EEG averages for the N400 spatio-temporal window are normally distributed.
2. Observations depend *completely* on the participant. (Participants have nothing in common.)
3. There is a linear relationship between cloze and the EEG signal for the trial.

What likelihood and priors can we choose here?

[1] For simplicity, we assume that they share the same standard deviation.

5.1.2.2 Likelihood and priors

The likelihood is a normal distribution as before:

$$signal_n \sim Normal(\alpha_{i[n]} + c\_cloze_n \cdot \beta_{i[n]}, \sigma) \tag{5.3}$$

This model is actually a collection of models: one linear model for each participant, with a single standard deviation σ across all participants.[2]

The priors are as follows:

$$\begin{aligned} \alpha_i &\sim Normal(0, 10) \\ \beta_i &\sim Normal(0, 10) \\ \sigma &\sim Normal_+(0, 50) \end{aligned} \tag{5.5}$$

As before, n represents each observation, that is, the nth row in the data frame, which has N rows. The notation $i[n]$, which follows follows Gelman and Hill (2007), identifies the participant index; for example, if $i[10] = 3$, then the 10th row of the data-frame is from participant 3.

In `brms`, such a model can be fit by removing the common intercept with `0 +`. This is an idiosyncratic syntax within `brms` (it departs from the `lme4` conventions), and ensure that one intercept and one slope is estimated for *each* level of `subject`. The intercept corresponds to `factor(subject)` and the slope to `c_cloze:factor(subject)`. This means that the model fits 37 independent intercepts and 37 independent slopes. By setting a prior to `class = b` and omitting `coef`, we are essentially setting identical priors to all the intercepts and slopes of the model. Notice that the parameters are independent of each other, it's only our previous knowledge about their possible values (encoded in the priors) that is identical. We can set different priors to each intercept and slope, but that will mean to set 74 priors!

[2] Incidentally, the `lmList` function in the `lme4` package fits a frequentist variant of the above model with the code specification `lmList(n400 ~ c_cloze|subject,data = df_eeg_data)`. The only difference here from our model is that a separate standard deviation σ is fit for each participant:

$$signal_n \sim Normal(\alpha_{i[n]} + c\_cloze_n \cdot \beta_{i[n]}, \sigma_{i[n]}) \tag{5.4}$$

```
fit_N400_np <- brm(n400 ~ 0 + subject + c_cloze:subject,
                   prior =
                       c(prior(normal(0, 10), class = b),
                         prior(normal(0, 50), class = sigma)),
                   data = df_eeg_data)
```

For this model, printing a summary means printing the 75 parameters ($\alpha_{1,...,37}$, $\beta_{1,...,37}$, and σ). We could do this as always by printing out the model results: just type `fit_N400_np`. Instead, one can plot $\beta_{1,..,37}$ using `bayesplot`. (`brms` also a includes a wrapper to this function called `stanplot`). We can peek at the internal names that `brms` gives to the parameters with `parnames(fit_N400_np)`; they are `b_subject`, then the subject index and then `:c_cloze`. The code below changes the subject labels back to their original numerical indices and plots them in Figure 5.2. The subjects are ordered by the magnitude of their mean effects.

The model M_{np} does not estimate a unique population-level effect; instead, that there is a different effect estimated for each subject. However, given the posterior means from each subject, it is still possible to calculate the average of these estimates $\hat{\beta}_{1,...,n}$:

```
## choose parameters:
ind_effects_np <- paste0("b_subject",unique(df_eeg_data$subject), ":c_cloze")
average_beta_across_subj <- posterior_samples(fit_N400_np,
                                              pars=ind_effects_np) %>%
    rowMeans()

grandmean<-c(mean=mean(average_beta_across_subj),
  quantile(average_beta_across_subj, c(.025,.975)))
```

The 95% credible interval of this overall mean effect is plotted in Figure 5.2 as two vertical lines.

```
## reorder plot by magnitude of mean:
dat<-mcmc_intervals_data(fit_N400_np,point_est="mean")
dat<-dat[38:74,]
```

```
dat<-dat[order(dat$m),]
## strip unnecessary characters from subject ids:
dat$subj<-stringr::str_replace(dat$parameter,"b_subject","")
dat$subj<-stringr::str_replace(dat$subj,":c_cloze","")
dat$subj<-factor(dat$subj,
                 levels=unique(dat$subj))

mcmc_intervals(fit_N400_np, pars=as.character(dat$parameter),
               prob = 0.8,
               prob_outer = 0.95,
               point_est = "mean") +
#theme(axis.text.y = element_blank())+
  ylab("subject")+
    scale_y_discrete(name="subjects",
               labels=as.character(unique(dat$subj)))+
  xlab("microvolts")+
geom_vline(xintercept = grandmean[2],colour="black")+
geom_vline(xintercept = grandmean[3],colour="black")
```

5.1.3 Varying intercept and varying slopes model (M_v)

One major problem with the no-pooling model is that it's not modeling the assumption that there is an overally effect and that participants have estimates that vary around that mean effect. To obtain an estimate of the overall effect, we just averaged the estimates of the mean subject-level effects. The model can be modified to explicitly assume that the subjects have an overall effect common to all the subjects, with the individual subjects deviating from this common effect.

Assuming that there is an overall effect that is common to the subjects will result in the estimation of posteriors for each participant being also influenced by what we know about all the subjects together. We'll first fit a hierarchical model with uncorrelated varying intercept and slope.[3]

[3] An analogous frequentist model can be fit with `lmer` from the package `lme4`, using `(c_cloze||subj)` for the random effects.

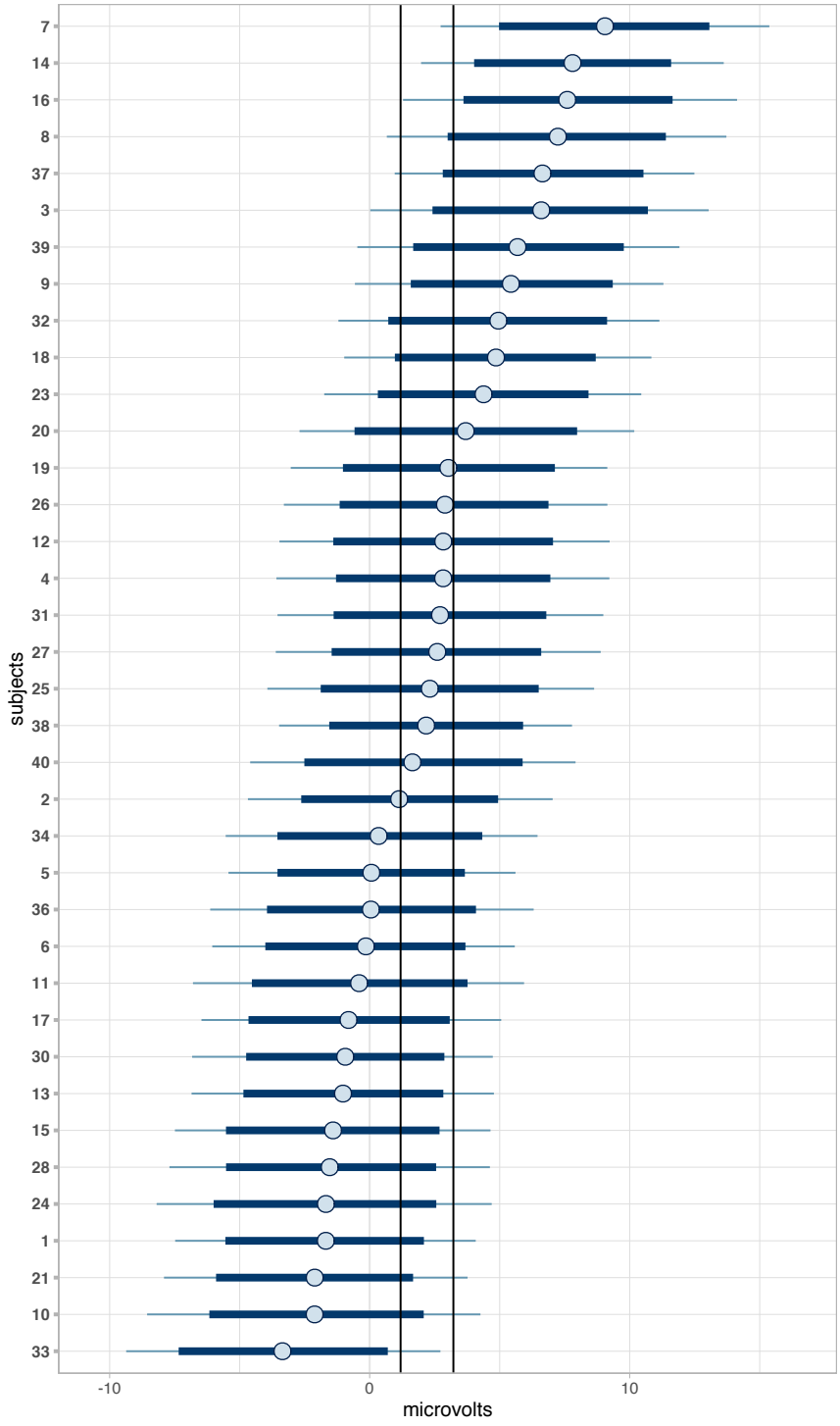


FIGURE 5.2: 95% credible intervals of the effect of Cloze probability for each subject according to the no pooling model.

5.1.3.1 Model assumptions

1. EEG averages for the N400 spatio-temporal window are normally distributed.
2. All subjects share a grand mean signal voltage and of the mean effect of predictability.
3. Each subject deviates to some extent (this is made precise below) from the grand mean and from the mean effect of predictability. This implies that there is some between-subject variability in the individual-level intercept and slope adjustments by subject.
4. There is a linear relationship between cloze and the EEG signal.

5.1.3.2 Likelihood and priors

The likelihood now incorporates an assumption that both the intercept and slope are adjusted by participant.

$$signal_n \sim Normal(\alpha + u_{0,i[n]} + c\_cloze_n \cdot (\beta + u_{1,i[n]}), \sigma) \quad (5.6)$$

$$\begin{aligned} \alpha &\sim Normal(0, 10) \\ \beta &\sim Normal(0, 10) \\ u_0 &\sim Normal(0, \tau_{u_0}) \\ u_1 &\sim Normal(0, \tau_{u_1}) \\ \tau_{u_0} &\sim Normal_+(0, 20) \\ \tau_{u_1} &\sim Normal_+(0, 20) \\ \sigma &\sim Normal_+(0, 50) \end{aligned} \quad (5.7)$$

In this model each subject has their own intercept adjustment, $u_{0,i}$, and slope adjustment, $u_{1,i}$. If $u_{0,i}$ is positive, the subject will have a more positive EEG signal than the grand mean average. If $u_{1,i}$ is positive, the subject will have a more positive EEG response to a change of one unit in `c_cloze` than the overall mean effect. The parameters u are sometimes called random effects and thus a model with fixed effects (α and β) and random effects is called a mixed model. However, random effects have different meanings in different contexts. To avoid ambiguity, `brms` calls

these parameters *group-level* effects. Notice that since we are estimating α and u at the same time and we assume that the average of the u's is 0 (since it is assumed to be normally distributed with mean 0), what is common between the subjects, the grand mean, is estimated as the intercept α, and the deviations of individual subjects' means from this grand mean are the adjustments u_0. Similarly, the mean effect of cloze is estimated as β, and the deviations of individual subjects' mean effects of cloze from β are the adjustment u_1. The standard deviations of these two adjustment terms, τ_{u_0} and τ_{u_1}, respectively, represent between participant variability; see also Box 5.1.

Thus, the model M_v has three *standard deviations*: σ, τ_{u_0} and τ_{u_1}. In statistics, it is conventional to talk about variances (the square of these standard deviations); for this reason, these standard deviations are also (confusingly) called *variance components*. The variance components τ_{u_0} and τ_{u_1} characterize between-subject variability, and the variance component σ characterizes within-subject variability.

Notice that the by-subject adjustments u_0 and u_1 are parameters in the model, and therefore have priors defined on them. Parameters that appear in the prior specifications for parameters, such as τ_u, are often called *hyperparameters*, and the priors on such hyperparameters are called *hyperpriors*. Thus, the parameter u_0 has $Normal(0, \tau_{u_0})$ as a prior; τ_{u_0} is a hyperparameter, and the hyperprior on τ_{u_0} is $Normal(0, 20)$.[4]

In general, the standard deviations for the by-subject adjustments are smaller than the standard deviation of the observations (which is the within-subjects standard deviation). That is, usually the between-subject variability in the intercepts and slopes is smaller than the within-subjects variability in the data. For this reason, reducing the scale of the truncated normal distribution to 20 (in comparison to 50) seems reasonable for the priors of the τ parameters. As always, we can do a sensitivity analysis to verify that our priors are reasonably uninformative (if we intended them to be uninformative).

[4] One could in theory keep going deeper and deeper, definer hyper-hyperpriors etc., but the model would quickly become impossible to fit.

Box 5.1. ***Some important (and sometimes confusing) points:***

- Why does u have a mean of 0?

  Because we want u to capture only differences between subjects, we could achieve the same by assuming the following relationship between the likelihood and the intercept and slope:

$$\begin{aligned} signal_n &\sim Normal(\alpha_{i[n]} + \beta_{i[n]} \cdot c\_cloze_n, \sigma) \\ \alpha_i &\sim Normal(\alpha, \tau_{u_0}) \\ \beta_i &\sim Normal(\beta, \tau_{u_1}) \end{aligned} \tag{5.8}$$

  And in fact, that's another common way to write the model.

- Why do the adjustments u have a normal distribution?

  Mostly because of convention, that's the way it's implemented in most frequentist mixed models. But also because if we don't know anything about the distribution besides its mean and variance, the normal distribution is the most conservative assumption (see also chapter 9 of McElreath, 2015).

For now, we are assuming that there is no relationship (no correlation) between the by-subject intercept and slope adjustments u_0 and u_1; as in `lmer`, this lack of correlation is indicated using in `brms` using the double pipe `||`. In `brms`, we need to specify hyperpriors for τ_{u_0} and τ_{u_1}; these are called `sd` in `brms`, to distinguish these standard deviations from σ. As with the population-level effects, the by-subjects intercept adjustments are implicitly fit for the group-level effects and thus `(c_cloze || subject)` is equivalent to `(1 + c_cloze || subject)`. If we don't want an intercept we need to explicitly indicate it with `(0 + c_cloze || subject)` or `(-1 + c_cloze || subject)`. Such a removal of the intercept is not normally done.

```
fit_N400_v <- brm(n400 ~ c_cloze + (c_cloze || subject),
                  prior =
                      c(prior(normal(0, 10), class = Intercept),
```

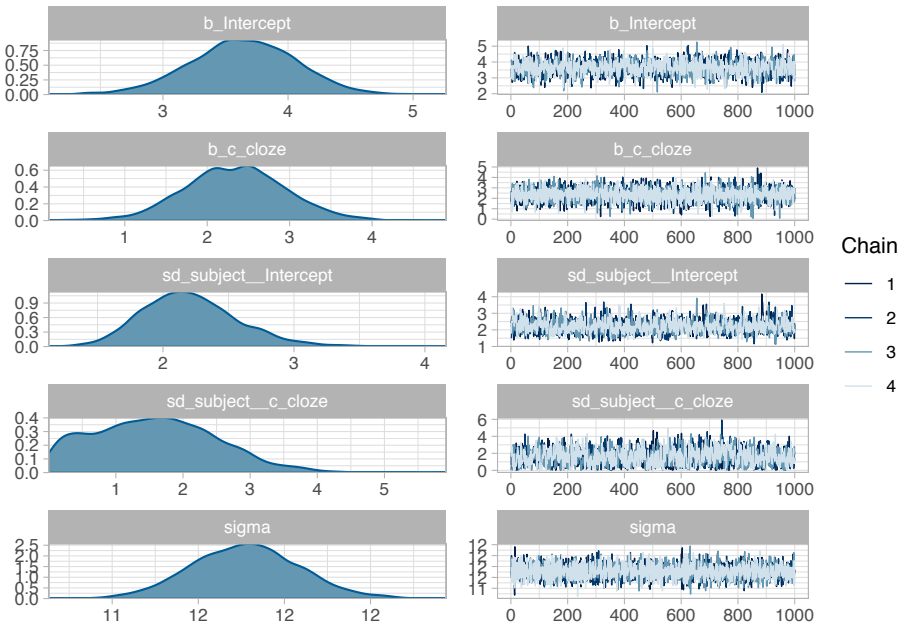
```
                        prior(normal(0, 10), class = b, coef = c_cloze),
                        prior(normal(0, 50), class = sigma),
                        prior(normal(0, 20), class = sd, coef = Intercept, group = subject),
                        prior(normal(0, 20), class = sd, coef = c_cloze, group = subject)
                        ),
              data = df_eeg_data)
```

When we print a `brms` fit, we first see the summaries of the posteriors of the standard deviation of the by-group intercept and slopes, τ_{u_0} and τ_{u_1} as `sd(Intercept)` and `sd(c_cloze)`, and then, as with previous models, the population-level effects, α and β as `Intercept` and `c_cloze`, and the scale of the likelihood, σ, as `sigma`.

```
posterior_summary(fit_N400_v)
```

Because the above command will result in pages of output, it is easier to understand the summary graphically:

```
plot(fit_N400_v, N=6)
```



Because we estimated how the population-level effect of Cloze is adjusted for each subject, we could examine how each subject is being affected by the manipulation. For this we do the following, and we plot it in Figure 5.3. Notice that these are adjustments, $u_{1,1,\ldots,37}$, and not the effect of the manipulation by subject, $\beta + u_{1,1,\ldots,37}$.

```
## reorder plot by magnitude of mean:
dat_v<-mcmc_intervals_data(fit_N400_v,point_est="mean")
dat_v<-dat_v[43:79,]
dat_v<-dat_v[order(dat_v$m),]
## strip unnecessary characters from subject ids:
dat_v$subj<-stringr::str_replace(dat_v$parameter,"r_subject\\[","")
dat_v$subj<-stringr::str_replace(dat_v$subj,",c_cloze\\]","")
dat_v$subj<-factor(dat_v$subj,
                   levels=unique(dat_v$subj))

mcmc_intervals(fit_N400_v, pars=as.character(dat_v$parameter),
               prob = 0.8,
               prob_outer = 0.95,
               point_est = "mean") +
#theme(axis.text.y = element_blank())+
  ylab("subject")+
    scale_y_discrete(name="subjects",
               labels=as.character(unique(dat_v$subj)))+
  xlab("microvolts")
```

There is an important difference between the no-pooling model and the varying intercepts and slopes model we just fit. The no-pooling model fits each individual subject's intercept and slope independently for each subject. By contrast, the varying intercepts and slopes model takes *all* the subjects' data into account in order to compute the fixed effects α and β; and the model shrinks the by-subject intercept and slope adjustments towards the fixed effects estimates. We can see the shrinkage of the estimates in the varying intercepts model when we compare them with the estimates of the no pooling model (M_{np}) in Figure 5.4.

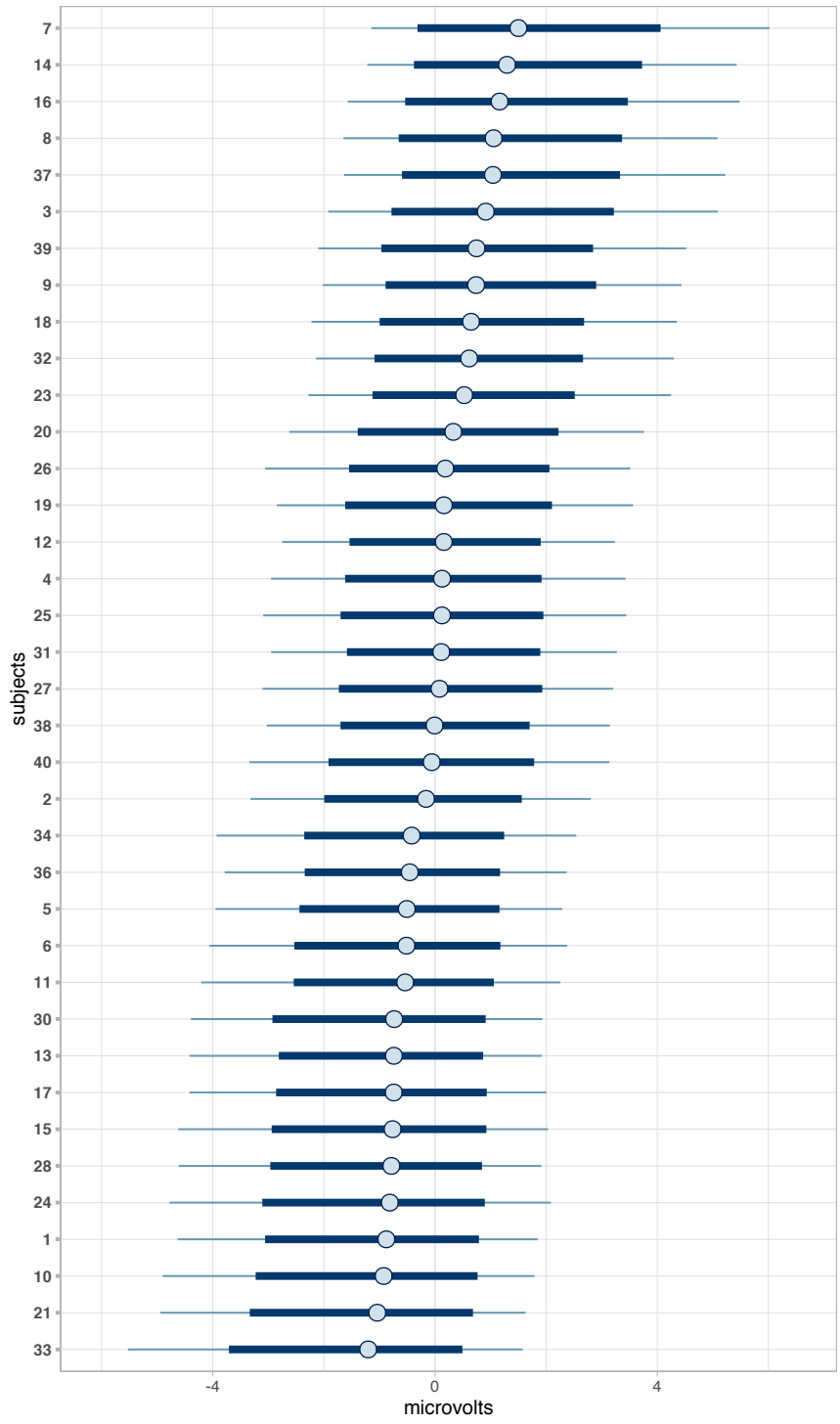


FIGURE 5.3: 95% credible intervals of adjustments to the effect of Cloze probability for each subject ($u_{1,1..37}$) according to the varying intercept and varying slopes model.

```
# We'll need to make the plot "manually"
# No pooling model
ind_effects_v <- paste0("r_subject[",unique(df_eeg_data$subject), ",c_cloze]")

par_np <- posterior_summary(fit_N400_np)[ind_effects_np,] %>%
    as_tibble() %>%
    mutate(model = "No pooling",
           subj = unique(df_eeg_data$subject))
# For the hierarchical model is more complicated,
# because we want the effect (beta) + adjustment:
par_h <- posterior_samples(fit_N400_v) %>%
    select(ind_effects_v)  %>%
    # We create a dataframe where each column is beta + u_{1,i}
    mutate_all( ~ . + posterior_samples(fit_N400_v)$b_c_cloze) %>%
    # We iterate over each column and create a dataframe with
    # estimate and the 95% CI of each iteration:
    map_dfr(~ tibble(Estimate = mean(.),
                Q2.5 = quantile(.,.025),
                Q97.5 = quantile(., .975))) %>%
    # We add a column to identify that the model,
    # and one with the subject labels:
    mutate(model = "Hierarchical",
           subj = unique(df_eeg_data$subject))
# The mean and 95% CI of both models in one dataframe:
by_subj_df <- bind_rows(par_h, par_np)

by_subj_df<-by_subj_df[order(by_subj_df$Estimate),]

ggplot(by_subj_df,
       aes(ymin = Q2.5, ymax = Q97.5,x=subj, y = Estimate, color=model,
           shape = model)) +
    geom_errorbar(position = position_dodge(1)) +
    geom_point(position = position_dodge(1)) +
    # We'll also add the mean and 95% CrI of the overall difference to the plot:
    geom_hline(yintercept = posterior_summary(fit_N400_v)["b_c_cloze","Estimate"]) +
    geom_hline(yintercept = posterior_summary(fit_N400_v)["b_c_cloze","Q2.5"],
```

```
            linetype = "dotted",size = .5)+
  geom_hline(yintercept = posterior_summary(fit_N400_v)["b_c_cloze","Q97.5"],
            linetype = "dotted",size = .5) +
  xlab("N400 effect of predictability") +
  coord_flip()
```

5.1.4 Correlated varying intercept varying slopes model (M_h)

The model M_v allowed for differences in intercept (mean voltage) and slopes (effects of Cloze) across subjects, but it has the implicit assumption that these are independent. It is in principle possible that subjects showing more negative voltage may also show stronger effects (or weaker effects). Next, we fit a model that assumes a correlation between the intercepts and slopes. We model the correlation between varying intercepts and slopes, by defining a variance-covariance matrix Σ between the by-subject varying intercepts and slopes, and by assuming that both adjustments (intercept and slope) come from a multivariate (in this case, a bivariate) normal distribution. See Box 5.2 for a short overview of the essential details regarding the variance-covariance matrix.

Box 5.2. ***The variance-covariance matrix and the corresponding correlation matrix:***

The variances in a multivariate distribution will be composed of

- variances for each random variable
- covariances between pairs of random variables, which includes some correlation ρ between pairs of random variables

E.g., for a bivariate distribution with random variables u_0 and u_1, this information is expressed in a so-called variance-covariance matrix.

$$\Sigma_u = \begin{pmatrix} \tau^2_{u_0} & \rho_u \tau_{u_0} \tau_{u1} \\ \rho \tau_{u_0} \tau_{u_1} & \tau^2_{u_1} \end{pmatrix} \tag{5.9}$$

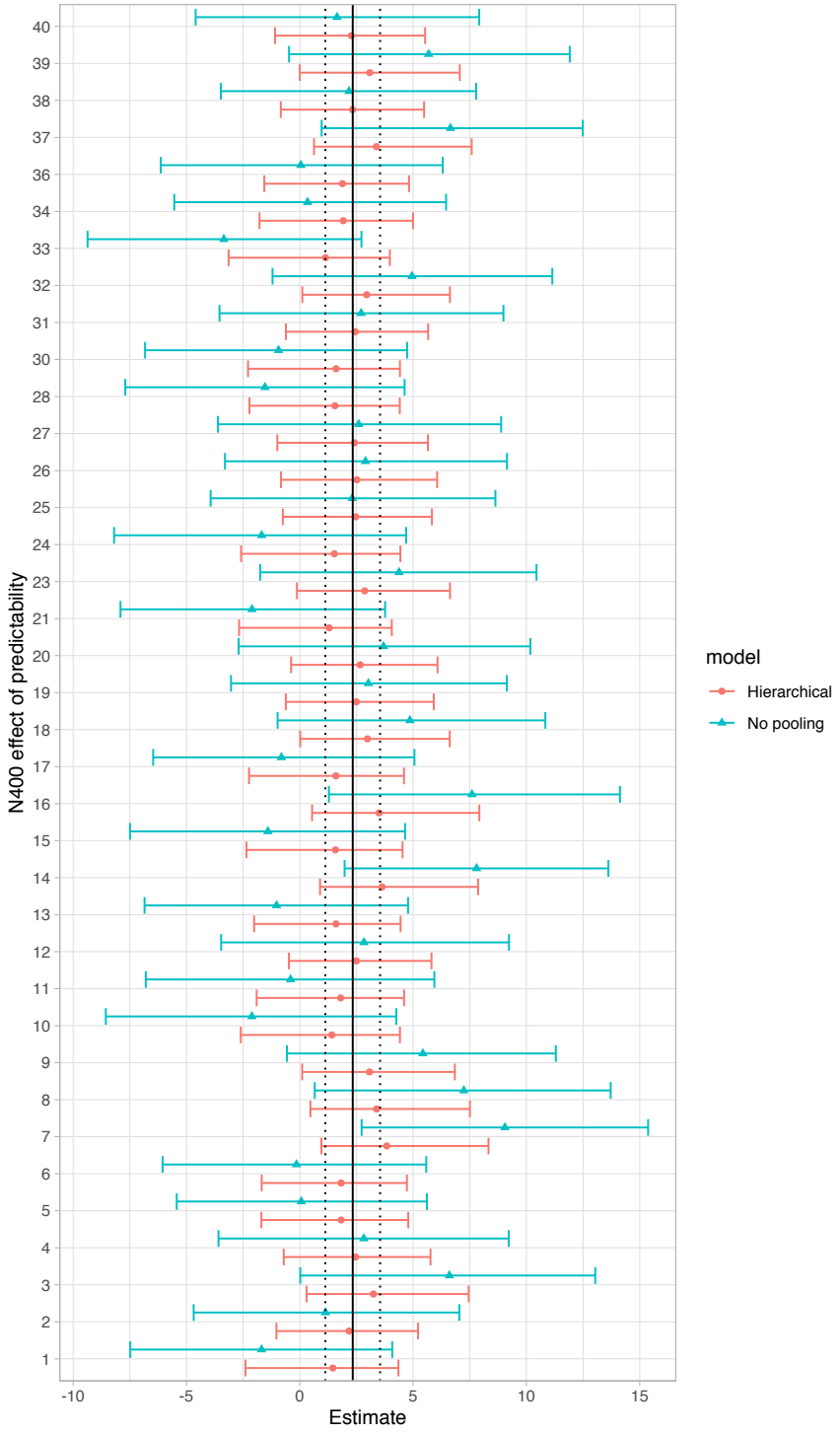


FIGURE 5.4: Comparison of the estimates of effect of Cloze probability for each subject between the no pooling and the varying intercept and varying slopes, hierarchical, model.

the covariance $Cov(u_0, u_1)$ between two variables X and Y is defined as the product of their correlation ρ_u and their standard deviations τ_{u_0} and τ_{u_1}, such that, $Cov(u_0, u_1) = \rho_u \tau_{u_0} \sigma_{u_1}$.

The covariance matrix can be decomposed into a matrix of standard deviations and a correlation matrix. For our example, the correlation matrix looks like this:

$$_u = \begin{pmatrix} 1 & \rho_u \\ \rho_u & 1 \end{pmatrix} \tag{5.10}$$

This means that we can decompose the covariance matrix into three parts:

$$\Sigma_u = \begin{pmatrix} \tau_{u_0} & 0 \\ 0 & \tau_{u_1} \end{pmatrix} \begin{pmatrix} 1 & \rho_u \\ \rho_u & 1 \end{pmatrix} \begin{pmatrix} \tau_{u_0} & 0 \\ 0 & \tau_{u_1} \end{pmatrix} \tag{5.11}$$

The importance of the correlation matrix becomes is that a prior will be defined on the correlation matrix rather than on the individual correlation parameter. One reason for this is generality: in more complex designs, such as $2 \times 2 \times 2$ factorial experiments, the variance covariance matrix is much larger than in our example in the text, but the proliferation of correlations that result is no problem for `brms` or Stan because we define the prior on the correlation matrix.

- In M_h, we model the EEG data with the following assumptions:

    1. EEG averages for the N400 spatio-temporal window are normally distributed.
    2. Some aspects of the mean signal voltage and of the effect of predictability depend on the participant, and these two might be correlated, i.e., we assume group-level intercepts, and slopes, and a correlation between them by-subject.
    3. There is a linear relationship between cloze and the EEG signal for the trial.

The likelihood remains identical to the model without a correlation between group-level intercepts and slopes (section 5.1.3):

$$signal_n \sim Normal(\alpha + u_{i[n],0} + c\_cloze_n \cdot (\beta + u_{i[n],1}), \sigma) \quad (5.12)$$

The correlation is indicated in the priors on the adjustments for intercept $u_{,0}$ and slopes $u_{,1}$.

- Priors:

$$\begin{aligned} \alpha &\sim Normal(0, 10) \\ \beta &\sim Normal(0, 10) \\ \sigma &\sim Normal_+(0, 50) \\ \begin{pmatrix} u_{i,0} \\ u_{i,1} \end{pmatrix} &\sim \mathcal{N}\left(\begin{pmatrix} 0 \\ 0 \end{pmatrix}, \Sigma_u\right) \end{aligned} \quad (5.13)$$

In this model, we define an $n \times 2$ matrix $\mathbf{u}$ as coming from a biivariate normal distribution with a variance-covariance matrix Σ_u. This matrix has the variances of the adjustment to the intercept and to the slope respectively along the diagonal, and the covariances on the off-diagonal (lower and upper triangles). The covariance $Cov(u_0, u_1)$ between two variables u_0 and u_1 is defined as the product of their correlation ρ and their standard deviations τ_{u_0} and τ_{u_1}, such that, $Cov(u_0, u_2) = \rho_u \tau_{u_0} \tau_{u_1}$.

$$\Sigma_u = \begin{pmatrix} \tau^2_{u_0} & \rho_u \tau_{u_0} \tau_{u_1} \\ \rho_u \tau_{u_0} \tau_{u_1} & \tau^2_{u_1} \end{pmatrix} \quad (5.14)$$

In order to specify a prior for Σ_u, we need priors for the standard deviations, τ_{u_0}, and τ_{u_1}, and also for their correlation, ρ_u. We can use the same priors for τ as before. For the correlation parameter ρ_u (and the correlation matrix more generally), we use the so-called LKJ prior. The basic idea of the LKJ correlation distribution is that as its parameter (usually called *eta*, η, here is 2) increases, the prior increasingly concentrates around the unit correlation matrix (i.e., favors less correlation: ones in the diagonals and values close to zero in the lower and upper triangles). At $\eta = 1$, the LKJ correlation distribution is uninformative (similar to $Beta(1, 1)$), at $\eta < 1$, it favors extreme correlations (similar to $Beta(a < 1, b < 1)$). We set $\eta = 2$ so that we don't favor extreme correlations and we still represent our lack of knowledge. Figure 5.5 shows a visualization of different parametrizations of the LKJ prior.

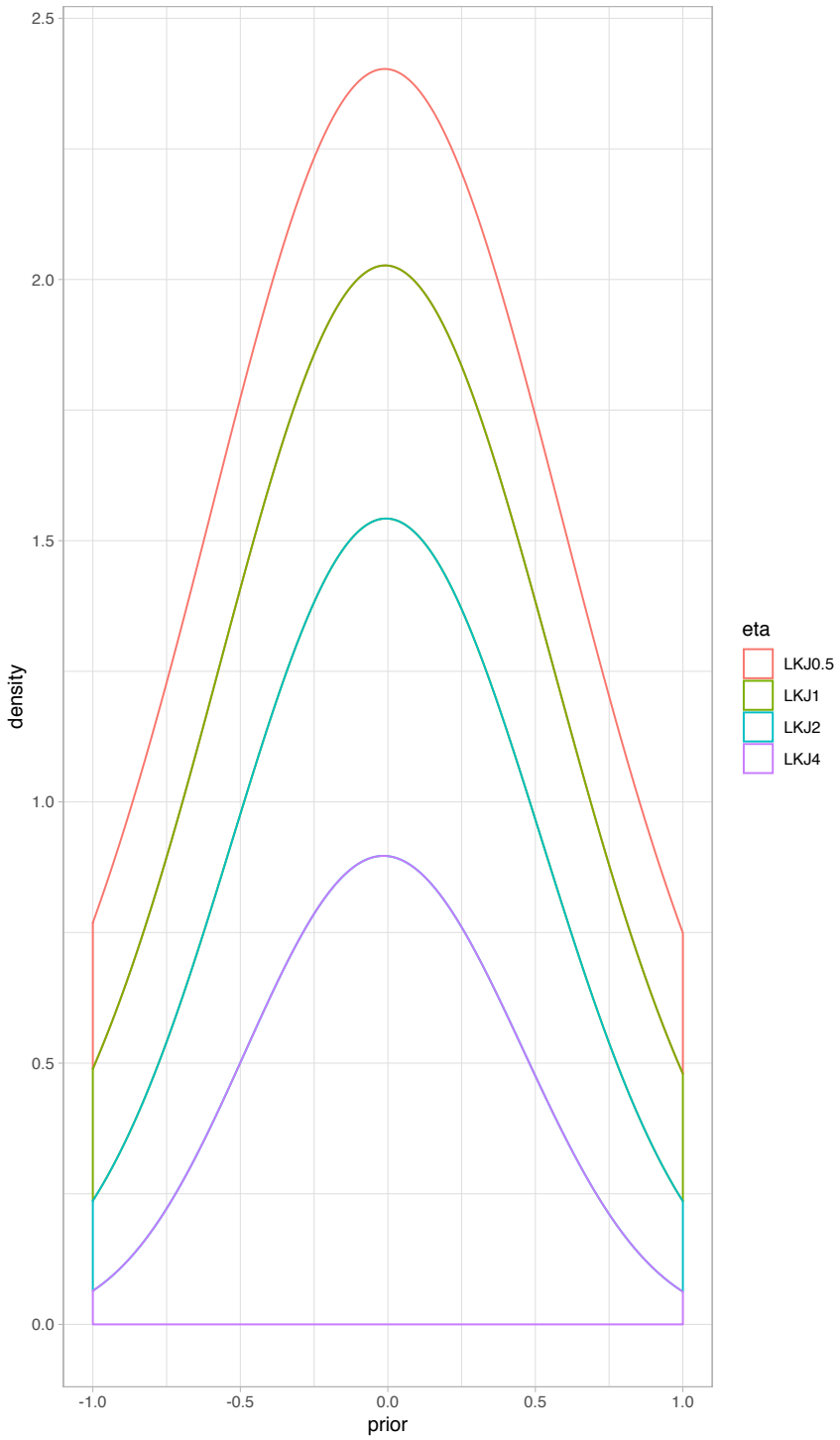


FIGURE 5.5: Visualization of the LKJ prior with four different values of the η parameter.

$$\begin{aligned} \tau_{u_0} &\sim Normal_+(0, 20) \\ \tau_{u_1} &\sim Normal_+(0, 20) \\ \rho_u &\sim LKJcorr(2) \end{aligned} \tag{5.15}$$

We indicate in our `brms` model that we assume a possible correlation between the by-subject intercept and slope with the single pipe `|`. As before the intercept is implicitly fit. This means that we need to add a new priors for the correlation, ρ_u, `cor` in `brms`.

```
fit_N400_h <- brm(n400 ~ c_cloze + (c_cloze | subject),
                  prior =
                      c(prior(normal(0, 10), class = Intercept),
                        prior(normal(0, 10), class = b, coef = c_cloze),
                        prior(normal(0, 50), class = sigma),
                        prior(normal(0, 20), class = sd, coef = Intercept, group = subject),
                        prior(normal(0, 20), class = sd, coef = c_cloze, group = subject),
                        prior(lkj(2), class = cor, group= subject)),
              data = df_eeg_data)
```

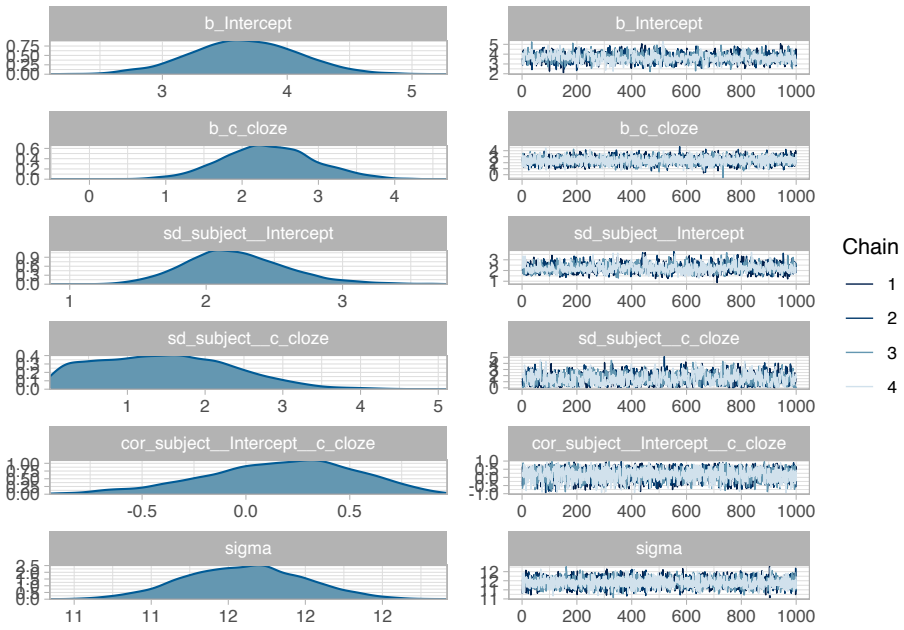
The estimates do not change much in comparison with the varying intercept/slope model, probably because the estimation of the correlation is quite poor (i.e., there is a lot of uncertainty).

```
fit_N400_h
```

```
##  Family: gaussian
##   Links: mu = identity; sigma = identity
## Formula: n400 ~ c_cloze + (c_cloze | subject)
##    Data: df_eeg_data (Number of observations: 2827)
## Samples: 4 chains, each with iter = 2000; warmup = 1000; thin = 1;
##          total post-warmup samples = 4000
##
## Group-Level Effects:
## ~subject (Number of levels: 37)
##                         Estimate Est.Error l-95% CI
## sd(Intercept)               2.22      0.38     1.56
```

```
## sd(c_cloze)                1.47      0.87     0.09
## cor(Intercept,c_cloze)     0.16      0.36    -0.62
##                         u-95% CI Rhat Bulk_ESS Tail_ESS
## sd(Intercept)              3.07 1.00     1317     2073
## sd(c_cloze)                3.25 1.00     1108     1655
## cor(Intercept,c_cloze)     0.78 1.00     3769     2608
##
## Population-Level Effects:
##           Estimate Est.Error l-95% CI u-95% CI Rhat
## Intercept     3.66      0.44     2.79     4.52 1.00
## c_cloze       2.35      0.61     1.20     3.57 1.00
##           Bulk_ESS Tail_ESS
## Intercept     1127     1800
## c_cloze       3791     2829
##
## Family Specific Parameters:
##       Estimate Est.Error l-95% CI u-95% CI Rhat
## sigma    11.64      0.15    11.34    11.94 1.00
##       Bulk_ESS Tail_ESS
## sigma     4906     2960
##
## Samples were drawn using sampling(NUTS). For each parameter, Eff.Sample
## is a crude measure of effective sample size, and Rhat is the potential
## scale reduction factor on split chains (at convergence, Rhat = 1).
```

```
plot(fit_N400_h, N=6)
```



We are now half-way to the so-called maximal hierarchical model (Barr et al., 2013), because everything that we said about subjects is also relevant for items. The next section spells out this type of model.

5.1.5 By-subjects and by-items correlated varying intercept varying slopes model (M_{sih})

Our new model, M_{sih} will allow for differences in intercept (mean voltage) and slopes (effects of predictability) across subject *and* across items. Here we assume a possible correlation between varying intercepts and slopes by subjects, and another one by items.

- In M_{sih}, we model the EEG data with the following assumptions:

  1. EEG averages for the N400 spatio-temporal window are normally distributed.
  2. Some aspects of the mean signal voltage and of the effect of predictability depend on the participant, i.e., we assume group-level intercepts, and slopes, and a correlation between them by-subject.
  3. Some aspects of the mean signal voltage and of the effect of pre-

dictability depend on the item, i.e., we assume group-level intercepts, and slopes, and a correlation between them by-item.
4. There is a linear relationship between cloze and the EEG signal for the trial.

- Likelihood:

$$signal_n \sim Normal(\alpha + u_{i[n],0} + w_{[j[n]],0} + c\_cloze_n \cdot (\beta + u_{i[n],1} + w_{j[n],1}), \sigma) \tag{5.16}$$

- Priors:

$$\begin{aligned}
\alpha &\sim Normal(0, 10) \\
\beta &\sim Normal(0, 10) \\
\sigma &\sim Normal_+(0, 50) \\
\begin{pmatrix} u_{i,0} \\ u_{i,1} \end{pmatrix} &\sim \mathcal{N}\left(\begin{pmatrix} 0 \\ 0 \end{pmatrix}, \Sigma_u\right) \\
\begin{pmatrix} w_{i,0} \\ w_{i,1} \end{pmatrix} &\sim \mathcal{N}\left(\begin{pmatrix} 0 \\ 0 \end{pmatrix}, \Sigma_w\right)
\end{aligned} \tag{5.17}$$

We have added the index j, which represents each item, as we did with subjects; $j[n]$ indicates the item that corresponds to the observation n.

We have hyperpriors as before:

$$\begin{aligned}
\Sigma_u &= \begin{pmatrix} \tau_{u_0}^2 & \rho_u \tau_{u_0} \tau_{u_1} \\ \rho_u \tau_{u_0} \tau_{u_1} & \tau_{u_1}^2 \end{pmatrix} \\
\Sigma_w &= \begin{pmatrix} \tau_{w_0}^2 & \rho_w \tau_{w_0} \tau_{w_1} \\ \rho_w \tau_{w_0} \tau_{w_1} & \tau_{w_1}^2 \end{pmatrix}
\end{aligned} \tag{5.18}$$

$$\begin{aligned}
\tau_{u_0} &\sim Normal_+(0, 20) \\
\tau_{u_1} &\sim Normal_+(0, 20) \\
\rho_u &\sim LKJcorr(2) \\
\tau_{w_0} &\sim Normal_+(0, 20) \\
\tau_{w_1} &\sim Normal_+(0, 20) \\
\rho_w &\sim LKJcorr(2)
\end{aligned} \tag{5.19}$$

We set identical priors to by-items group-level effects as to the by-subject

ones, because we don't have different prior information about them. However, bear in mind that the estimation for items is completely independent from the estimation for subjects. Although we wrote many more equations than before, the `brms` model is quite straightforward to extend:

```
fit_N400_sih <- brm(n400 ~ c_cloze + (c_cloze | subject) + (c_cloze |item ),
                    prior =
                        c(prior(normal(0, 10), class = Intercept),
                          prior(normal(0, 10), class = b, coef = c_cloze),
                          prior(normal(0, 50), class = sigma),
                          prior(normal(0, 20), class = sd, coef = Intercept, group = subject),
                          prior(normal(0, 20), class = sd, coef = c_cloze, group = subject),
                          prior(lkj(2), class =cor, group = subject),
                          prior(normal(0, 20), class = sd, coef = Intercept, group = item),
                          prior(normal(0, 20), class = sd, coef = c_cloze, group = item),
                          prior(lkj(2), class =cor, group = item)),
                    data = df_eeg_data)
```

We can also simplify the call to `brms`, when we assign the same priors to the by-subject and by-item parameters:

```
fit_N400_sih <- brm(n400 ~ c_cloze + (c_cloze | subject) + (c_cloze |item ),
                    prior =
                        c(prior(normal(0, 10), class = Intercept),
                          prior(normal(0, 10), class = b),
                          prior(normal(0, 50), class = sigma),
                          prior(normal(0, 20), class = sd),
                          prior(lkj(2), class =cor)),
                    data = df_eeg_data)
```

We have new group-level effects in the summary, but again the estimate of the effect of Cloze remains virtually unchanged.

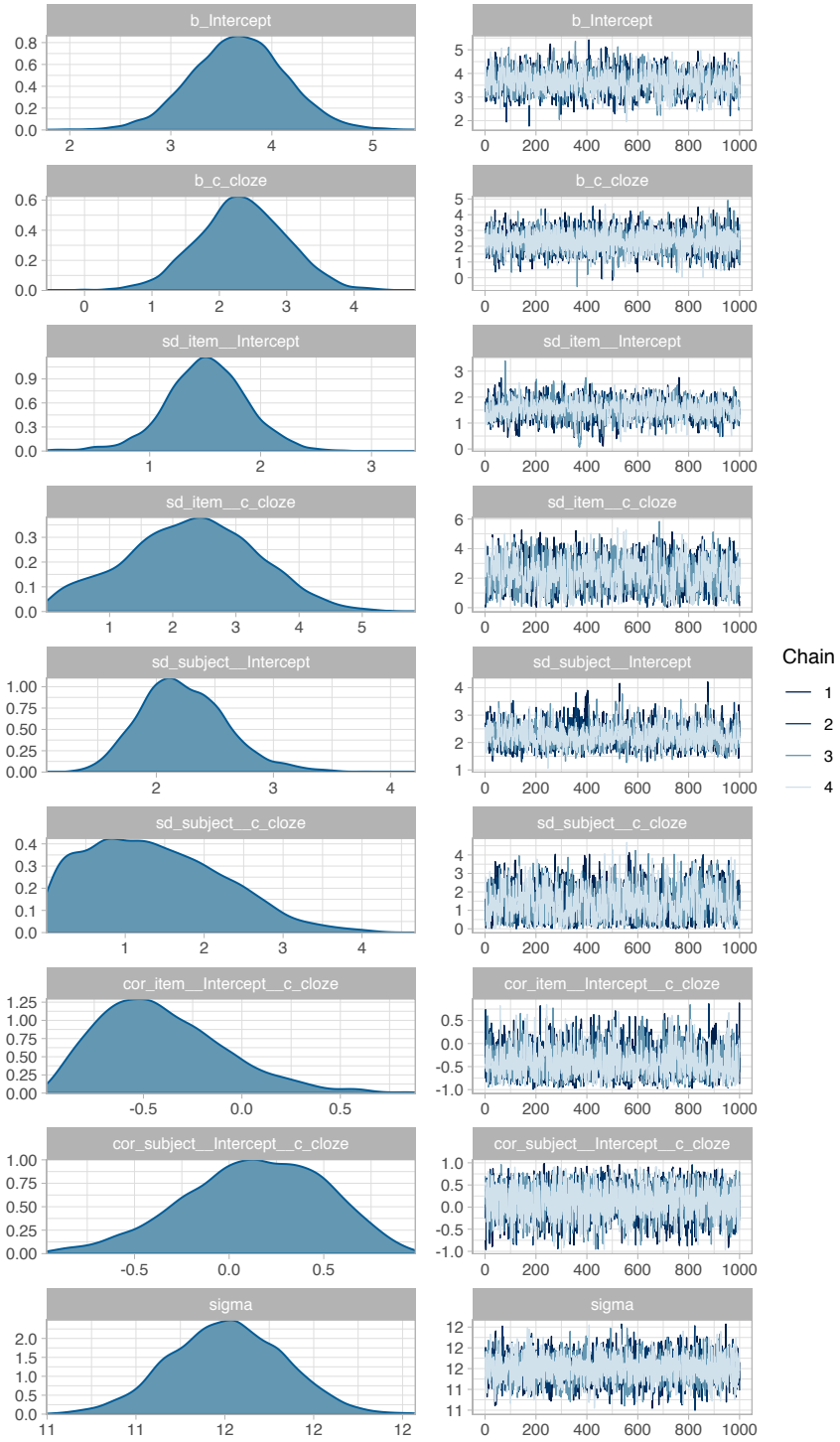
```
fit_N400_sih
```

```
##  Family: gaussian
##   Links: mu = identity; sigma = identity
```

```
## Formula: n400 ~ c_cloze + (c_cloze | subject) + (c_cloze | item)
##    Data: df_eeg_data (Number of observations: 2827)
## Samples: 4 chains, each with iter = 2000; warmup = 1000; thin = 1;
##          total post-warmup samples = 4000
##
## Group-Level Effects:
## ~item (Number of levels: 80)
##                         Estimate Est.Error l-95% CI
## sd(Intercept)               1.51      0.37     0.76
## sd(c_cloze)                 2.29      1.03     0.30
## cor(Intercept,c_cloze)     -0.41      0.32    -0.90
##                         u-95% CI Rhat Bulk_ESS Tail_ESS
## sd(Intercept)               2.22 1.00     1158     1065
## sd(c_cloze)                 4.27 1.00      942      945
## cor(Intercept,c_cloze)      0.32 1.00     1894     1995
##
## ~subject (Number of levels: 37)
##                         Estimate Est.Error l-95% CI
## sd(Intercept)               2.24      0.37     1.59
## sd(c_cloze)                 1.37      0.86     0.07
## cor(Intercept,c_cloze)      0.14      0.37    -0.64
##                         u-95% CI Rhat Bulk_ESS Tail_ESS
## sd(Intercept)               3.08 1.00     1502     2540
## sd(c_cloze)                 3.19 1.00     1185     1647
## cor(Intercept,c_cloze)      0.78 1.00     4039     2491
##
## Population-Level Effects:
##           Estimate Est.Error l-95% CI u-95% CI Rhat
## Intercept     3.67      0.46     2.75     4.59 1.00
## c_cloze       2.34      0.67     1.02     3.63 1.00
##           Bulk_ESS Tail_ESS
## Intercept     1815     2341
## c_cloze       4224     3095
##
## Family Specific Parameters:
##       Estimate Est.Error l-95% CI u-95% CI Rhat
## sigma    11.51      0.16    11.19    11.81 1.00
```

```
##       Bulk_ESS Tail_ESS
## sigma     5887     2729
##
## Samples were drawn using sampling(NUTS). For each parameter, Eff.Sample
## is a crude measure of effective sample size, and Rhat is the potential
## scale reduction factor on split chains (at convergence, Rhat = 1).
```

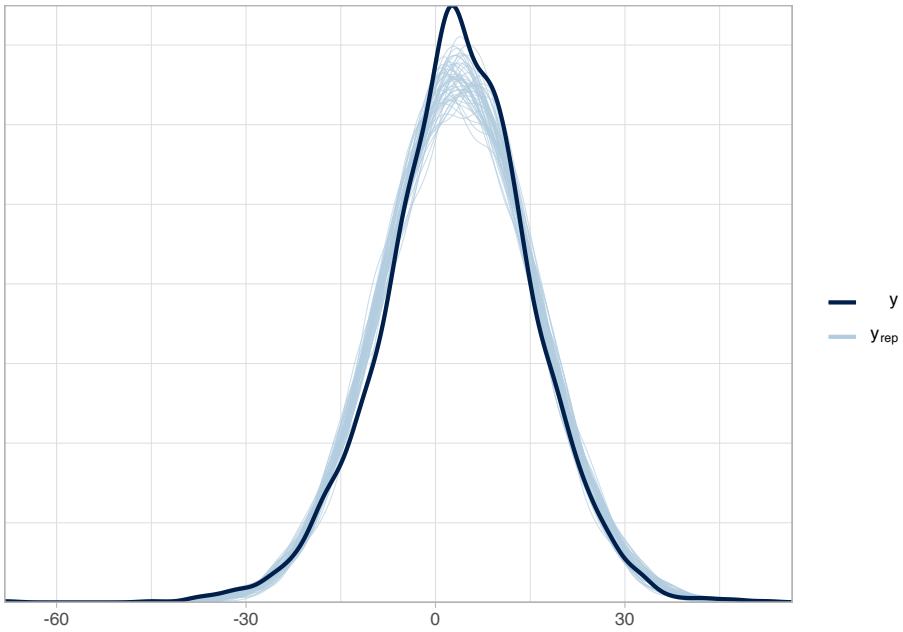
```
plot(fit_N400_sih, N=9)
```


b_Intercept
b_c_cloze
sd_item__Intercept
sd_item__c_cloze
sd_subject__Intercept
sd_subject__c_cloze
cor_item__Intercept__c_cloze
cor_subject__Intercept__c_cloze
sigma
Chain
1
2
3
4

5.1.6 Beyond the so-called maximal models–Distributional regression models

We can use posterior predictive checks to verify that our last model can capture the entire signal distribution.

```
pp_check(fit_N400_sih, nsamples =50, type="dens_overlay")
```



However, we know that in ERP studies, large levels of impedance between the recording electrodes and the skin tissue increase the noise in the recordings (**?**) . Given that skin tissue is different between subjects, it could be the case that the level of noise varies by participant. It might be a good idea to verify that our model is good enough for capturing the by-subject data pattern. We adapt the code from 4.1.4 and we plot it in Figure 5.6.

```
df_eeg_pred <- posterior_predict(fit_N400_sih,
                                 nsamples = 1000) %>%
    array_branch(margin = 1) %>%
    map_dfr( function(yrep_iter) {
        df_eeg_data %>%
```

```
            mutate(n400 = yrep_iter)
    }, .id = "iter") %>%
    mutate(iter = as.numeric(iter))

df_eeg_pred %>% filter(iter < 100) %>%
    ggplot(aes(n400, group=iter)) +
    geom_line(alpha = .05, stat="density", color = "blue") +
    geom_density(data=df_eeg_data, aes(n400),
                 inherit.aes = FALSE, size =1)+
    facet_wrap(subject ~ .) +
    xlab("Signal in the N400 spatiotemporal window")
```

Figure 5.6 hints that we might be misfitting some subjects. Another approach to examine whether we misfit the by-subject noise level is to plot posterior distributions of the standard deviations and compared them with the observed standard deviation. This is achieved in the following code, and the result is shown in Figure 5.7. It is clear now that, for some subjects, the observed standard deviation lies outside the distribution of predictive standard deviations.

```
# predicted subject:
df_eeg_pred_summary <- df_eeg_pred %>%
    group_by(iter, subject) %>%
    summarize(sd = sd(n400))
# observed means:
df_eeg_summary <- df_eeg_data %>%
    group_by(subject) %>%
    summarize(sd = sd(n400, na.rm= TRUE))
# plot
ggplot(df_eeg_pred_summary, aes(sd)) +
    geom_histogram(alpha=.5)+
    geom_vline(aes(xintercept= sd),data= df_eeg_summary)+
    facet_wrap(subject ~.)+
    xlab("Standard deviation")
```

Why is our "maximal" hierarchical model misfitting the by-subject distri-

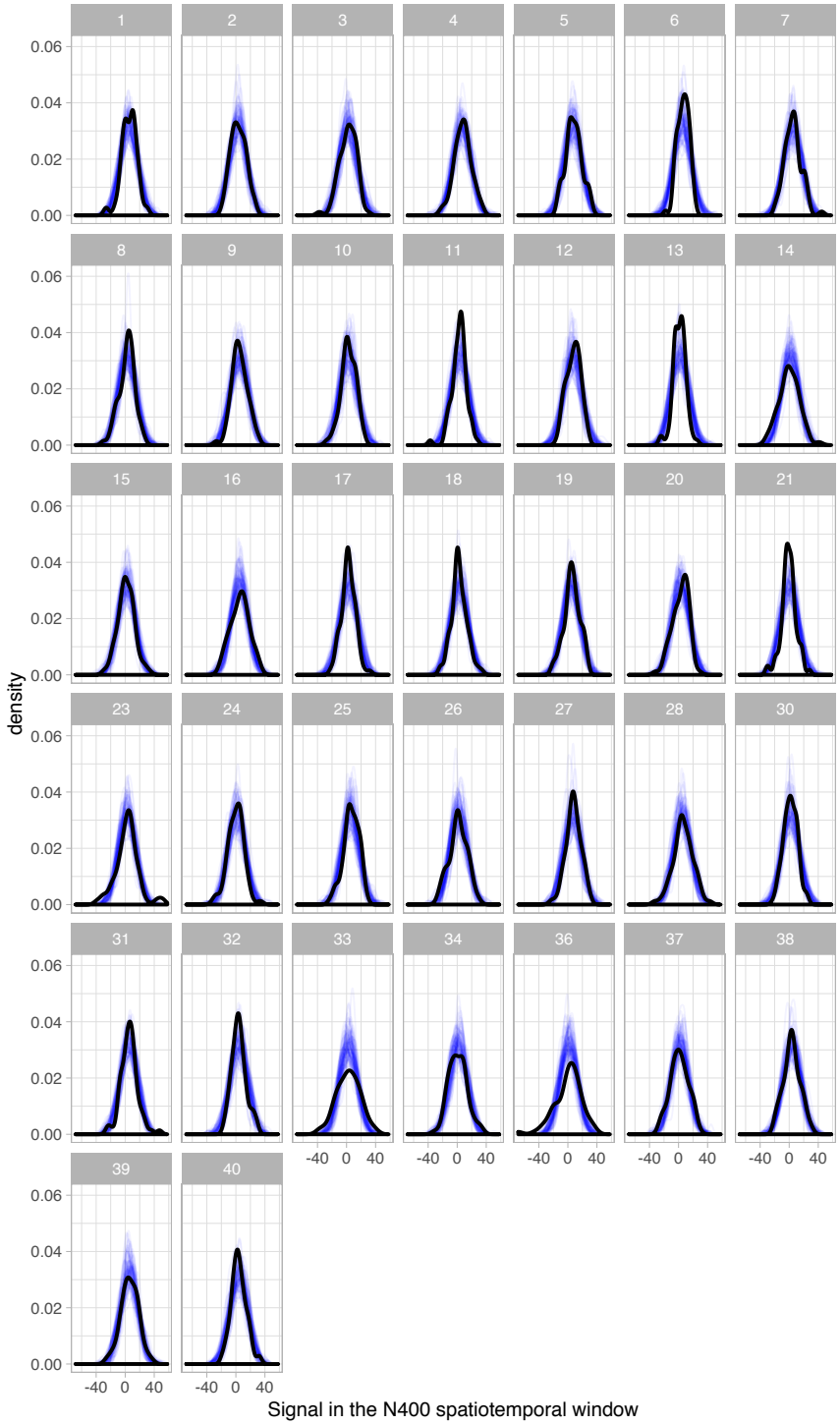


FIGURE 5.6: The plot shows 100 predicted distributions in blue density plots and the distribution of the average signal data in black density plots for the 37 subjects that participated in the experiment.

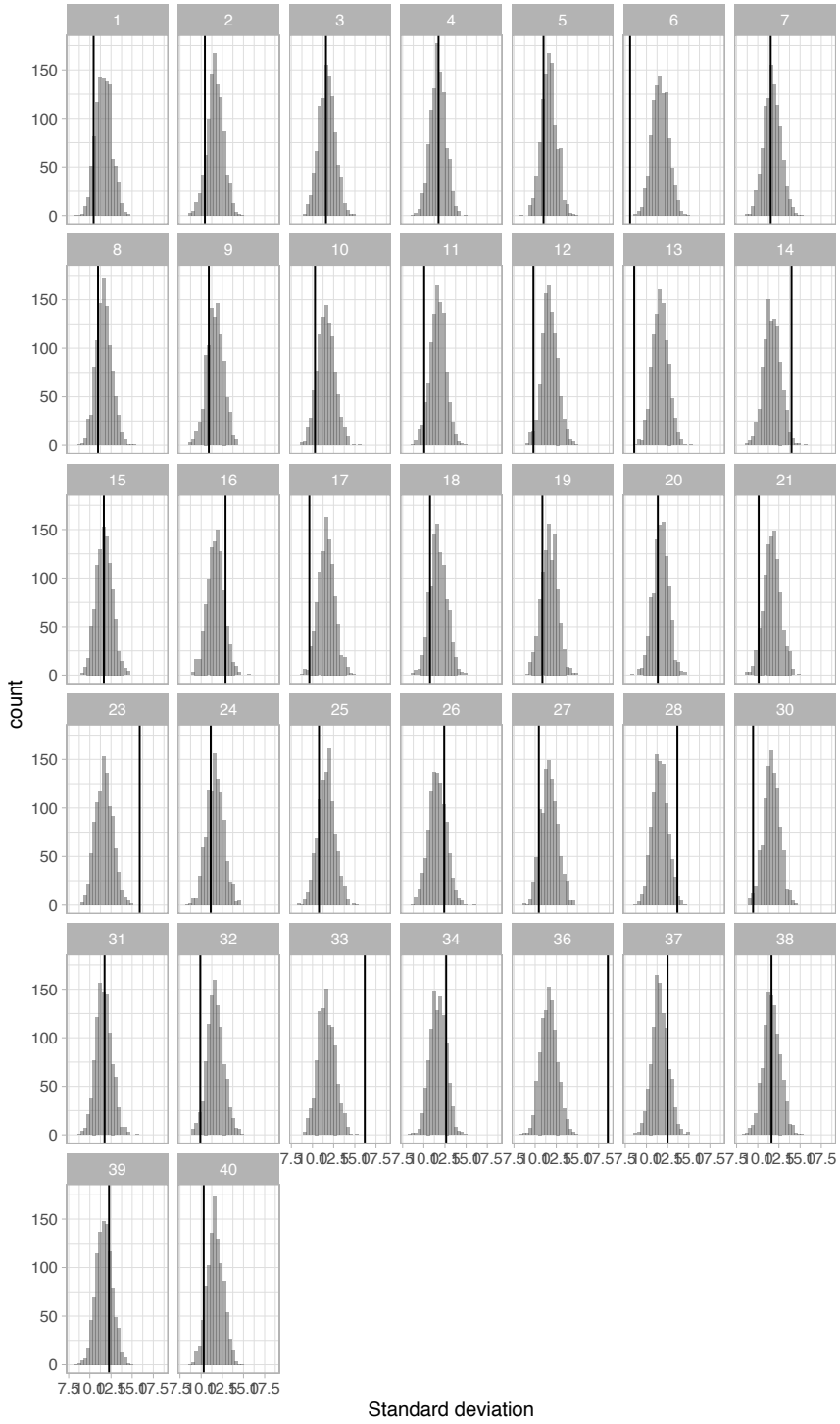


FIGURE 5.7: Distribution of posterior predicted standard deviations in gray and observed standard deviation in black lines by subject.

bution of data? This is because, the so-called maximal models are, in general and implicitly, models with the maximal group-level effect structure for the location parameter (e.g., the mean, μ, in a normal model). Other parameters (e.g., scale or shape parameters) are estimated as auxiliary parameters assuming them to be constant across observations and clusters. This assumption is so common that researchers may not be aware that it is just an assumption, which (in the Bayesian framework) can be changed. Changing this assumption leads to distributional regression models. These can be fit in `brms`.[5]

We are going to change our previous likelihood, so that the scale, σ has also a group-level effect structure. Notice that we exponentiate σ to make sure that the negative adjustments do not cause σ to become negative.

$$\begin{aligned} signal_n &\sim Normal(\alpha + u_{i[n],0} + w_{[j[n]],0} + c\_cloze_n \cdot (\beta + u_{i[n],1} + w_{j[n],1}), \sigma_n) \\ \sigma_n &= \exp(\sigma_\alpha + \sigma_{u_{i[n]}}) \end{aligned} \tag{5.20}$$

We just need to add priors to our new parameters (that replace the old prior for σ). We set the prior to the intercept of the standard deviation, σ_α, to be similar to our previous σ. For the variance component of σ, τ_{σ_u}, we set quite vague hyperpriors (recall that everything is exponentiated when it goes inside the likelihood).

$$\begin{aligned} \sigma_\alpha &\sim Normal(0, log(50)) \\ \sigma_u &\sim Normal(0, \tau_{\sigma_u}) \\ \tau_{\sigma_u} &\sim Normal_+(0, 5) \end{aligned} \tag{5.21}$$

This model can be fit in `brms` using the internal function `bf()`. This will allow us to set a hierarchical structure (and any regression) to the parameter σ. We also need to set new priors; these priors are idenfitied by `dpar = sigma`.

```
fit_N400_s <- brm(bf(n400 ~ c_cloze + (c_cloze | subject) + (c_cloze |item ),
                     sigma ~ 1+(1|subject)),
```

[5] https://web.archive.org/web/20191206093021/https://paul-buerkner.github.io/brms/articles/brms_distreg.html

```
                  prior =
                      c(prior(normal(0, 10), class = Intercept),
                        prior(normal(0, 10), class = b),
                        prior(normal(0, 20), class = sd),
                        prior(lkj(2), class =cor),
                        prior(normal(0,50), class = Intercept, dpar = sigma),
                        prior(normal(0,10), class = sd, group = subject,
                              dpar = sigma)
                        ),
                  data = df_eeg_data)
```

We inspect the output below, and we see that our estimate for the effect of Cloze remains very similar to our previous one.

```
posterior_summary(fit_N400_s)[3,c(1,3,4)]
```

```
## Estimate     Q2.5    Q97.5
##      2.3      1.0      3.6
```

Nonetheless, Figure 5.8 shows that the fit of the model with respect to the by-subject variability is much better than before. Furthermore, figure 5.9 shows that the observed standard deviations for each subject are well inside the posterior predictive distributions.

This raises the question of how much structure should we add to our statistical model. Should we assume that σ can also vary by items, and also by our experimental manipulation? Should we have a maximal model also for σ? Unfortunately, there are no clear answers that apply to every situation. The amount of complexity that we can introduce in a statistical model depends on (i) the answers we are looking for, that is, we should have the parameters that represent what we want to estimate, (ii) the size of the data at hand (more complex models require more data), (iii) our computing power; as the complexity increases models take increasingly long to converge and require more computer power to finish in a feasible time frame, and (iv) our domain and experimental knowledge.

Ultimately, all models are approximations (in the best case, when they are not plainly wrong) and we need to think carefully about which aspects

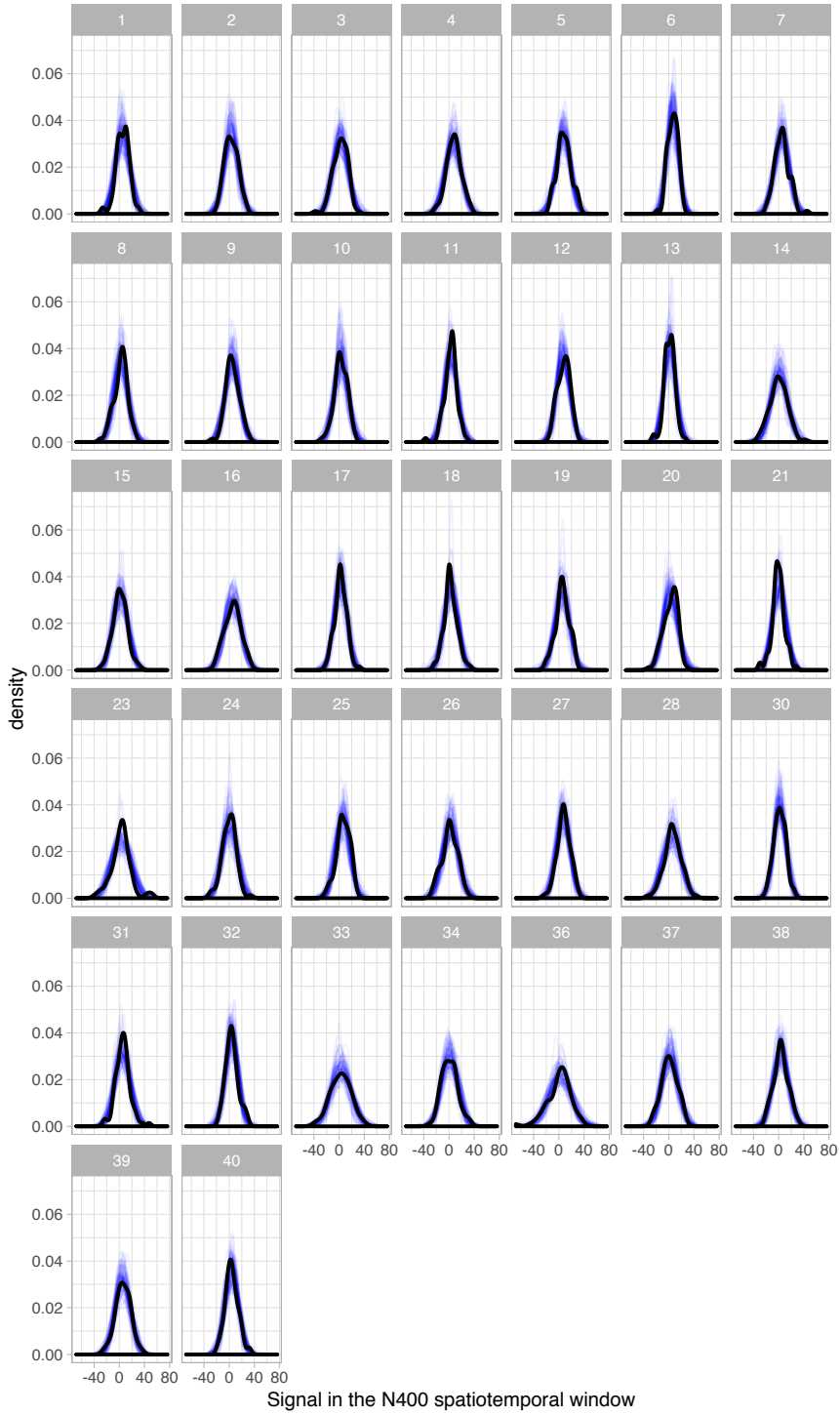


FIGURE 5.8: The plot shows 100 predicted distributions for the model that includes a hierarchical structure for σ in blue density plots and the distribution of the average signal data in black density plots for the 37 subjects that participated in the experiment.

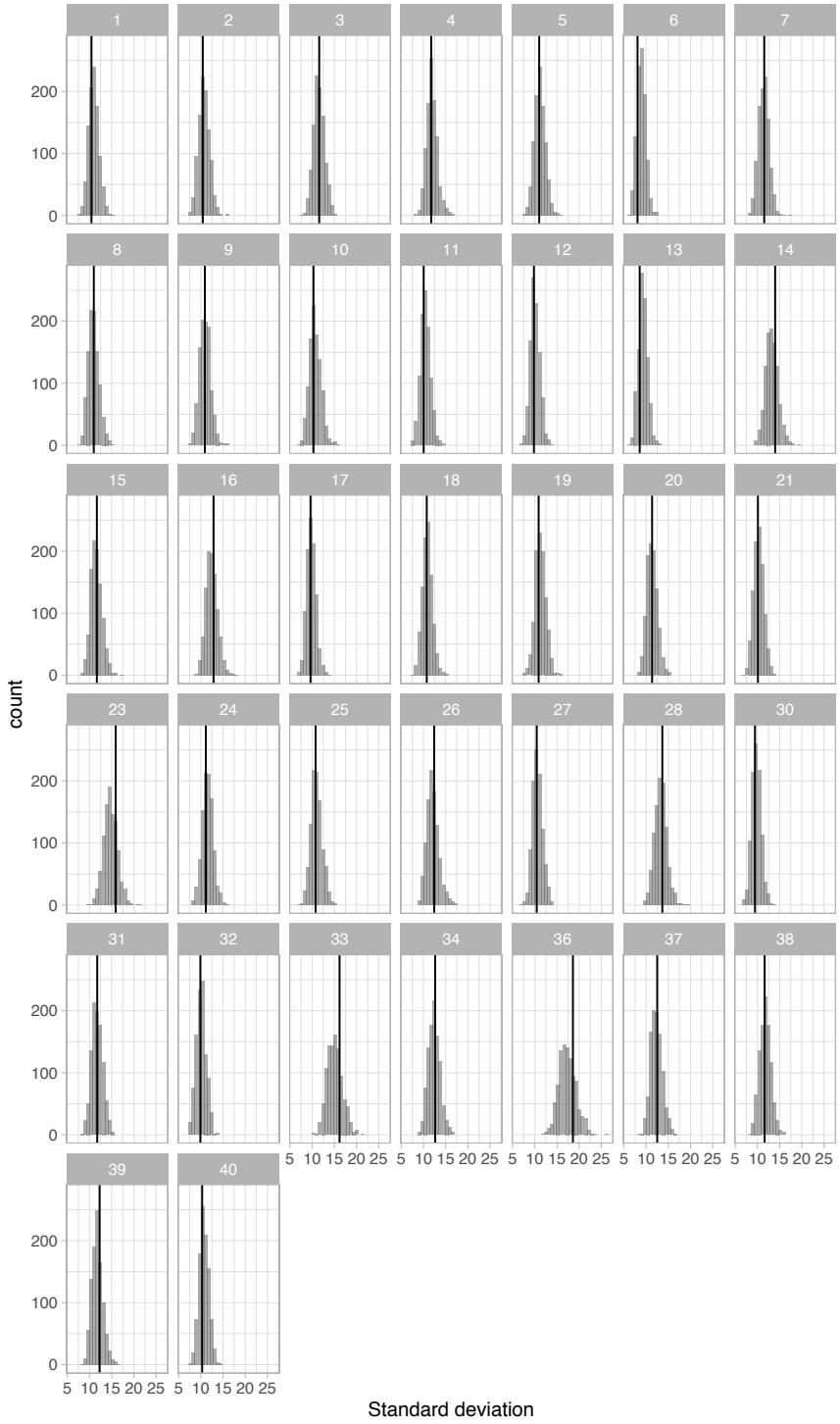


FIGURE 5.9: Distribution of posterior predicted standard deviations for the model that includes a hierarchical structure for σ in gray and observed standard deviation in black lines by subject.

of our data we have to account and which aspects we can abstract away from.

In the context of cognitive modeling, McClelland (2009) argues that models should not focus on a every single detail of the process they intend to explain. In order to understand a model, it needs to be simple enough. However, McClelland (2009) warns us that one must bear in mind that simplification does impact on what we can conclude from our analysis: A simplification can limit the phenomena that a model addresses, or can even lead to incorrect predictions. There is a continuum between purely statistical models (e.g., a linear regression) and computational cognitive models, that includes "hybrid" models such as the linear ballistic accumulator, where a great deal of cognitive detail is sacrificed for tractability. The conclusions of McClelland (2009) apply to any type of model in cognitive science: "Simplification is essential, but it comes at a cost, and real understanding depends in part on understanding the effects of the simplification".

5.2 Summary

TODO

5.2.1 Why should we take the trouble of fitting a Bayesian hierarchical model?

Carrying out Bayesian data analysis clearly requires much more effort than fitting a frequentist model: we have to define priors, verify that our model works, and decide how to interpret the results. By comparison, fitting a linear mixed model using `lme4` consists of only a single line of code; the model fit using `lmer` makes many assumptions, but they are hidden from the user. We want to emphasize that there are important motivations for fitting Bayesian hierarchical models.

- The same approach we used here can be used to extend any parameter of any model. This includes popular uses, such as logistic and Poisson regressions, and also useful models that are relatively rarely used in cognitive science such as multi-logistic regression (e.g., accuracy in

some task with more than two answers), ordered logistic (e.g., ratings), and models with a shifted log-normal distribution (see Nicenboim et al., 2016; Rouder, 2005). We provide examples of this flexibility in the coming chapters.

- Complex cognitive models can be extended hierarchically in a straightforward way, see Lee (2011) and Lee and Wagenmakers (2014). This is because, as we have seen with distributional regression models in section @ref{#sec:distrmodel}), any parameter can have a group-level effect strucure. Some examples of hierarchical computational cognitive models in psycholinguistics are Logačev and Vasishth (2016), Nicenboim and Vasishth (2018), Vasishth et al. (2017), and Vasishth et al. (2017).

5.3 Further reading

5.4 Exercises

3. Hierarchical model with a lognormal likelihood:

We begin with a classic question from the psycholinguistics literature: are subject relatives easier to process than object relatives? The data come from Experiment 1 in a paper by Grodner and Gibson (2005).

Scientific question: Is there a subject relative advantage in reading?

In two important papers, Gibson (2000) and Grodner and Gibson (2005) suggest that object relative clause sentences are more difficult to process than subject relative clause sentences because the distance between the relative clause verb *sent* and the head noun phrase of the relative clause, *reporter*, is longer in object vs subject relatives. Examples are shown below.

(1a) The *reporter* who the photographer *sent* to the editor was hoping for a good story. (ORC)

(1b) The *reporter* who *sent* the photographer to the editor was hoping for a good story. (SRC)

The underlying explanation has to do with memory processes: shorter linguistic dependencies are easier to process due to either reduced interference or decay, or both. For implemented computational models that spell this point out, see Lewis and Vasishth (2005) and Engelmann et al. (2018).

In the Grodner and Gibson data, the dependent measure is reading time at the relative clause verb, in milliseconds. We are expecting longer reading times in object gap sentences compared to subject gap.

```
gg05_data <- read_csv("data/GrodnerGibson2005E1.csv") %>%
    filter(item != 0) %>%
    mutate(word_positionnew = if_else(item != 15 &
                                          word_position > 10,
                                          word_position-1,
                                          word_position))
#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:
rc_data <- gg05_data %>% filter((condition == "objgap" & word_position == 6 ) |
            ( condition == "subjgap" & word_position == 4 ))
```

You should use a sum coding for the predictors. Here, object gaps are coded +1, subject gaps -1.

```
rc_data <- rc_data %>% mutate(ccond = if_else(condition == "objgap",  1,-1))
```

You should be able to now fit a maximal model (correlated varying intercept and slopes for subjects and items) assuming a lognormal likelihood, and examine the effect of relative clause attachment site (the predictor `ccond`) on reading times `rawRT`.

6

Contrast coding, interactions, etc

7

Important distributions

These distributions are used quite frequently in Bayesian data analyses, especially in psychology and linguistics applications. The Binomial and Poisson are discrete distributions, the rest are continuous. Each distribution comes with a family of `d-p-q-r` functions in R which allow us to compute the PDF/PMF, the CDF, the inverse CDF, and to generate random data. For example, the normal distribution's PDF is `dnorm`; the CDF and the inverse CDF are `pnorm` and `qnorm` respectively; and random data can be generated using `rnorm`.

The table below is adapted from `https://github.com/wzchen/probability_cheatsheet`, which is licensed under a Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International License.

to-do: check that the notation is consistent with the main text's.

Distribution	PMF/PDF and Support	Expected Value	Variance
Binomial $Binomial(n, \theta)$	$P(X = k) = \binom{n}{k}\theta^k(1-\theta)^{n-k}$ $k \in \{0, 1, 2, \ldots n\}$	$n\theta$	$n\theta(1-\theta)$
Poisson $Pois(\lambda)$	$P(X = k) = \frac{e^{-\lambda}\lambda^k}{k!}$ $k \in \{0, 1, 2, \ldots\}$	λ	λ
Uniform $Unif(a, b)$	$f(x) = \frac{1}{b-a}$ $x \in (a, b)$	$\frac{a+b}{2}$	$\frac{(b-a)^2}{12}$
Normal $Normal(\mu, \sigma)$	$f(x) = \frac{1}{\sigma\sqrt{2\pi}}e^{-\frac{(x-\mu)^2}{(2\sigma^2)}}$ $x \in (-\infty, \infty)$	$\mu = \frac{\sum_{i=1}^n x_i}{n}$	$\sigma^2 = \frac{\sum_{i=1}^n (x_i-\bar{x})^2}{n}$
Log-Normal $LogNormal(\mu, \sigma)$	$\frac{1}{x\sigma\sqrt{2\pi}}e^{-(\log x-\mu)^2/(2\sigma^2)}$ $x \in (0, \infty)$	$\theta = e^{\mu+\sigma^2/2}$	$\theta^2(e^{\sigma^2}-1)$
Beta Beta(a, b)	$f(x) = \frac{\Gamma(a+b)}{\Gamma(a)\Gamma(b)}x^{a-1}(1-x)^{b-1}$ $x \in (0, 1)$	$\mu = \frac{a}{a+b}$	$\frac{\mu(1-\mu)}{(a+b+1)}$
Exponential $Exp(\lambda)$	$f(x) = \lambda e^{-\lambda x}$ $x \in (0, \infty)$	$\frac{1}{\lambda}$	$\frac{1}{\lambda^2}$
Gamma $Gamma(a, \lambda)$	$f(x) = \frac{1}{\Gamma(a)}(\lambda x)^a e^{-\lambda x}\frac{1}{x}$ $x \in (0, \infty)$	$\frac{a}{\lambda}$	$\frac{a}{\lambda^2}$
Student-t $t(n)$ Cauchy is $t(1)$	$\frac{\Gamma((n+1)/2)}{\sqrt{n\pi}\Gamma(n/2)}(1+x^2/n)^{-(n+1)/2}$ $x \in (-\infty, \infty)$	0 if $n > 1$	$\frac{n}{n-2}$ if $n > 2$

8

Bayes factor: Definition

The Bayes factor is defined as follows. Given data y, the posterior odds of Model M0 is $p(M0|y)$, and the posterior odds of model M1 is $p(M1|y)$. Using the conditional probability rule, these can be expanded as:

$$p(M0|y) = \frac{p(y|M0)p(M0)}{p(y)} \text{ and } p(M1|y) = \frac{p(y|M1)p(M1)}{p(y)} \tag{8.1}$$

If we take the ratio of these two posterior odds, the denominator $p(y)$ cancels out giving us:

$$\frac{p(M0|y)}{p(M1|y)} = \frac{p(y|M0)p(M0)}{p(y|M1)p(M1)} \tag{8.2}$$

Rearranging terms on the right hand side, we can write:

$$\frac{p(M0|y)}{p(M1|y)} = \frac{p(y|M0)}{p(y|M1)}\frac{p(M0)}{p(M1)} \tag{8.3}$$

The term $\frac{p(y|M0)}{p(y|M1)}$ is the Bayes factor BF_{01}, and As the above equation shows, the Bayes factor is an updating factor: it updates the ratio of the prior odds $\frac{p(M0)}{p(M1)}$ to obtain the posterior odds.

Bibliography

Allaire, J., Xie, Y., McPherson, J., Luraschi, J., Ushey, K., Atkins, A., Wickham, H., Cheng, J., Chang, W., and Iannone, R. (2019). *rmarkdown: Dynamic Documents for R*. R package version 1.15.2.

Auguie, B. (2017). *gridExtra: Miscellaneous Functions for "Grid" Graphics*. R package version 2.3.

Aust, F. (2019). *citr: RStudio Add-in to Insert Markdown Citations*. R package version 0.3.2.

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., Mächler, M., Bolker, B., and Walker, S. (2015a). Fitting linear mixed-effects models using lme4. *Journal of Statistical Software*, 67(1):1–48.

Bates, D. and Maechler, M. (2019). *Matrix: Sparse and Dense Matrix Classes and Methods*. R package version 1.2-17.

Bates, D., Maechler, M., Bolker, B., and Walker, S. (2015b). Fitting linear mixed-effects models using lme4. *Journal of Statistical Software*, 67:1–48.

Blitzstein, J. K. and Hwang, J. (2014). *Introduction to probability*. Chapman and Hall/CRC.

Blumberg, E. J., Peterson, M. S., and Parasuraman, R. (2015). Enhancing multiple object tracking performance with noninvasive brain stimulation: a causal role for the anterior intraparietal sulcus. *Frontiers in Systems Neuroscience*, 9:3.

Buzsáki, G. and Mizuseki, K. (2014). The log-dynamic brain: How skewed distributions affect network operations. *Nature Reviews Neuroscience*, 15(4):264–278.

Bürkner, P.-C. (2019). *brms: Bayesian Regression Models using 'Stan'*. R package version 2.8.0.

Carpenter, B., Gelman, A., Hoffman, M. D., Lee, D., Goodrich, B., Betancourt,

M., Brubaker, M., Guo, J., Li, P., and Riddell, A. (2017). Stan: A probabilistic programming language. *Journal of Statistical Software*, 76(1).

Chang, W. (2018). *webshot: Take Screenshots of Web Pages*. R package version 0.5.1.

Cheng, J. (2018). *miniUI: Shiny UI Widgets for Small Screens*. R package version 0.1.1.1.

DeLong, K. A., Urbach, T. P., and Kutas, M. (2005). Probabilistic word preactivation during language comprehension inferred from electrical brain activity. *Nature Neuroscience*, 8(8):1117–1121.

Duane, S., Kennedy, A. D., Pendleton, B. J., and Roweth, D. (1987). Hybrid Monte Carlo. *Physics Letters B*, 195(2):216–222.

Eddelbuettel, D., Francois, R., Allaire, J., Ushey, K., Kou, Q., Russell, N., Bates, D., and Chambers, J. (2019). *Rcpp: Seamless R and C++ Integration*. R package version 1.0.2.

Engelmann, F., Jäger, L. A., and Vasishth, S. (2018). The effect of prominence and cue association in retrieval processes: A computational account. Manuscript submitted to Cognitive Science.

Fox, J. (2009). *A mathematical primer for social statistics*. Number 159. Sage.

Francois, R. (2017). *bibtex: Bibtex Parser*. R package version 0.4.2.

Frank, S. L., Otten, L. J., Galli, G., and Vigliocco, G. (2015). The ERP response to the amount of information conveyed by words in sentences. *Brain and Language*, 140:1–11.

Gabry, J. and Mahr, T. (2018). *bayesplot: Plotting for Bayesian Models*. R package version 1.6.0.

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.

Gelman, A. and Hill, J. (2007). *Data analysis using regression and multilevel/hierarchical models*. Cambridge University Press.

Gelman, A., Simpson, D., and Betancourt, M. (2017). The prior can often only be understood in the context of the likelihood. *Entropy*, 19(10):555.

Gibson, E. (2000). Dependency locality theory: A distance-based theory of linguistic complexity. In Marantz, A., Miyashita, Y., and O'Neil, W., editors, *Im-*

age, Language, Brain: Papers from the First Mind Articulation Project Symposium. MIT Press, Cambridge, MA.

Gill, J. (2006). *Essential mathematics for political and social research*. Cambridge University Press Cambridge.

Gohel, D., Wickham, H., Henry, L., and Ooms, J. (2019). *gdtools: Utilities for Graphical Rendering*. R package version 0.2.0.

Goodrich, B., Gabry, J., Ali, I., and Brilleman, S. (2018). rstanarm: Bayesian applied regression modeling via Stan. R package version 2.17.4.

Goodrich, B., Gelman, A., Carpenter, B., Hoffman, M., Lee, D., Betancourt, M., Brubaker, M., Guo, J., Li, P., Riddell, A., Inacio, M., Morris, M., Arnold, J., Goedman, R., Lau, B., Trangucci, R., Gabry, J., Kucukelbir, A., Grant, R., Tran, D., Malecki, M., and Gao, Y. (2019). *StanHeaders: C++ Header Files for Stan*. R package version 2.18.1-10.

Grodner, D. and Gibson, E. (2005). Consequences of the serial nature of linguistic input. *Cognitive Science*, 29:261–290.

Guo, J., Gabry, J., and Goodrich, B. (2019). *rstan: R Interface to Stan*. R package version 2.19.2.

Hayes, T. R. and Petrov, A. A. (2016). Mapping and correcting the influence of gaze position on pupil size measurements. *Behavior research methods*, 48(2):510–527.

Henry, L. and Wickham, H. (2019). *purrr: Functional Programming Tools*. R package version 0.3.2.

Hoffman, M. D. and Gelman, A. (2014). The No-U-Turn Sampler: Adaptively setting path lengths in Hamiltonian Monte Carlo. *Journal of Machine Learning Research*, 15(1):1593–1623.

Izrailev, S. (2014). *tictoc: Functions for timing R scripts, as well as implementations of Stack and List structures.* R package version 1.0.

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.

JASP Team (2019). JASP (Version 0.11.1)[Computer software].

Kolmogorov, A. N. (1933/2018). *Foundations of the Theory of Probability: Second English Edition*. Courier Dover Publications.

Kruschke, J. (2014). *Doing Bayesian data analysis: A tutorial with R, JAGS, and Stan*. Academic Press.

Kutas, M. and Federmeier, K. D. (2011). Thirty years and counting: Finding meaning in the N400 componentof the event-related brain potential (ERP). *Annual Review of Psychology*, 62(1):621–647.

Kutas, M. and Hillyard, S. A. (1980). Reading senseless sentences: Brain potentials reflect semantic incongruity. *Science*, 207(4427):203–205.

Kutas, M. and Hillyard, S. A. (1984). Brain potentials during reading reflect word expectancy and semantic association. *Nature*, 307(5947):161–163.

Lee, M. D. (2011). How cognitive modeling can benefit from hierarchical Bayesian models. *Journal of Mathematical Psychology*, 55(1):1–7.

Lee, M. D. and Wagenmakers, E.-J. (2014). *Bayesian cognitive modeling: A practical course*. Cambridge University Press.

Lewis, R. L. and Vasishth, S. (2005). An activation-based model of sentence processing as skilled memory retrieval. *Cognitive Science*, 29:1–45.

Limpert, E., Stahel, W. A., and Abbt, M. (2001). Log-normal Distributions across the Sciences: Keys and Clues. *BioScience*, 51(5):341.

Logačev, P. and Vasishth, S. (2016). A multiple-channel model of task-dependent ambiguity resolution in sentence comprehension. *Cognitive Science*, 40(2):266–298.

Lunn, D., Thomas, A., Best, N., and Spiegelhalter, D. (2000). WinBUGS-A Bayesian modelling framework: Concepts, structure, and extensibility. *Statistics and computing*, 10(4):325–337.

Mahowald, K., James, A., Futrell, R., and Gibson, E. (2016). A meta-analysis of syntactic priming in language production. *Journal of memory and language*, 91:5–27.

Mathot, S. (2018). Pupillometry: Psychology, physiology, and function. *Journal of Cognition*, 1(1):16.

McClelland, J. L. (2009). The place of modeling in Cognitive Science. *Topics in Cognitive Science*, 1(1):11–38.

McElreath, R. (2015). *Statistical rethinking: A Bayesian course with R examples*. Chapman and Hall/CRC.

Morin, D. J. (2016). *Probability: For the Enthusiastic Beginner*. Createspace Independent Publishing Platform.

Müller, K. and Walthert, L. (2019). *styler: Non-Invasive Pretty Printing of R Code*. R package version 1.1.1.

Neal, R. M. (2011). MCMC using Hamiltonian dynamics. In Brooks, S., Gelman, A., Jones, G., and Meng, X.-L., editors, *Handbook of Markov Chain Monte Carlo*, chapter 5. Taylor & Francis.

Nicenboim, B., Logačev, P., Gattei, C., and Vasishth, S. (2016). When high-capacity readers slow down and low-capacity readers speed up: Working memory and locality effects. *Frontiers in Psychology*, 7(280).

Nicenboim, B., Roettger, T. B., and Vasishth, S. (2018). Using meta-analysis for evidence synthesis: The case of incomplete neutralization in German. *Journal of Phonetics*, 70:39–55.

Nicenboim, B. and Vasishth, S. (2018). Models of retrieval in sentence comprehension: A computational evaluation using Bayesian hierarchical modeling. *Journal of Memory and Language*, 99:1–34.

Nieuwland, M. S., Politzer-Ahles, S., Heyselaar, E., Segaert, K., Darley, E., Kazanina, N., Von Grebmer Zu Wolfsthurn, S., Bartolozzi, F., Kogan, V., Ito, A., Mézière, D., Barr, D. J., Rousselet, G. A., Ferguson, H. J., Busch-Moreno, S., Fu, X., Tuomainen, J., Kulakova, E., Husband, E. M., Donaldson, D. I., Kohút, Z., Rueschemeyer, S.-A., and Huettig, F. (2018). Large-scale replication study reveals a limit on probabilistic prediction in language comprehension. *eLife*, 7.

Oberauer, K. (2019). Working memory capacity limits memory for bindings. *Journal of Cognition*, 2(1):40.

Oberauer, K. and Kliegl, R. (2001). Beyond resources: Formal models of complexity effects and age differences in working memory. *European Journal of Cognitive Psychology*, 13(1-2):187–215.

Plummer, M. (2016). Jags version 4.2.0 user manual.

R Core Team (2019). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria.

Ripley, B. (2019). *MASS: Support Functions and Datasets for Venables and Ripley's MASS*. R package version 7.3-51.4.

Rouder, J. N. (2005). Are unshifted distributional models appropriate for response time? *Psychometrika*, 70(2):377–381.

Royall, R. (1997). *Statistical Evidence: A likelihood paradigm*. Chapman and Hall, CRC Press, New York.

Salvatier, J., Wiecki, T. V., and Fonnesbeck, C. (2016). Probabilistic programming in python using PyMC3. *PeerJ Computer Science*, 2:e55.

Shiffrin, R., Lee, M., Kim, W., and Wagenmakers, E.-J. (2008). A survey of model evaluation approaches with a tutorial on hierarchical Bayesian methods. *Cognitive Science: A Multidisciplinary Journal*, 32(8):1248–1284.

Ulrich, R. and Miller, J. (1993). Information processing models generating lognormally distributed reaction times. *Journal of Mathematical Psychology*, 37(4):513–525.

Vaidyanathan, R., Xie, Y., Allaire, J., Cheng, J., and Russell, K. (2018). *htmlwidgets: HTML Widgets for R*. R package version 1.3.

Vasishth, S., Chen, Z., Li, Q., and Guo, G. (2013). Processing Chinese relative clauses: Evidence for the subject–relative advantage. *PLOS ONE*, 8(10):e77006.

Vasishth, S., Chopin, N., Ryder, R., and Nicenboim, B. (2017). Modelling dependency completion in sentence comprehension as a Bayesian hierarchical mixture process: A case study involving Chinese relative clauses. In *Proceedings of Cognitive Science Conference*.

Vasishth, S. and Engelmann, F. (2020). Sentence comprehension as a cognitive process: A computational approach. Under contract with Cambridge University Press.

Vasishth, S., Jaeger, L. A., and Nicenboim, B. (2017). Feature overwriting as a finite mixture process: Evidence from comprehension data. In *Proceedings of MathPsych/ICCM Conference*.

Wagenmakers, E.-J., Grasman, R. P. P. P., and Molenaar, P. C. M. (2005). On the relation between the mean and the variance of a diffusion model response time distribution. *Journal of Mathematical Psychology*, 49(3):195–204.

Wahn, B., Ferris, D. P., Hairston, W. D., and König, P. (2016). Pupil sizes scale with attentional load and task experience in a multiple object tracking task. *PLOS ONE*, 11(12):e0168087.

Wickham, H., Chang, W., Henry, L., Pedersen, T. L., Takahashi, K., Wilke, C., Woo, K., and Yutani, H. (2019a). *ggplot2: Create Elegant Data Visualisations Using the Grammar of Graphics*. R package version 3.2.0.

Wickham, H., François, R., Henry, L., and Müller, K. (2019b). *dplyr: A Grammar of Data Manipulation*. R package version 0.8.3.

Wickham, H. and Henry, L. (2019). *tidyr: Easily Tidy Data with 'spread()' and 'gather()' Functions*. R package version 0.8.3.

Wickham, H., Hester, J., and Francois, R. (2018). *readr: Read Rectangular Text Data*. R package version 1.3.1.

Wolodzko, T. (2019). *extraDistr: Additional Univariate and Multivariate Distributions*. R package version 1.8.11.

Xie, Y. (2019a). *bookdown: Authoring Books and Technical Documents with R Markdown*. R package version 0.13.2.

Xie, Y. (2019b). *knitr: A General-Purpose Package for Dynamic Report Generation in R*. R package version 1.25.

Xie, Y. (2019c). *servr: A Simple HTTP Server to Serve Static Files or Dynamic Documents*. R package version 0.15.

Xie, Y., Cheng, J., and Tan, X. (2018). *DT: A Wrapper of the JavaScript Library 'DataTables'*. R package version 0.5.